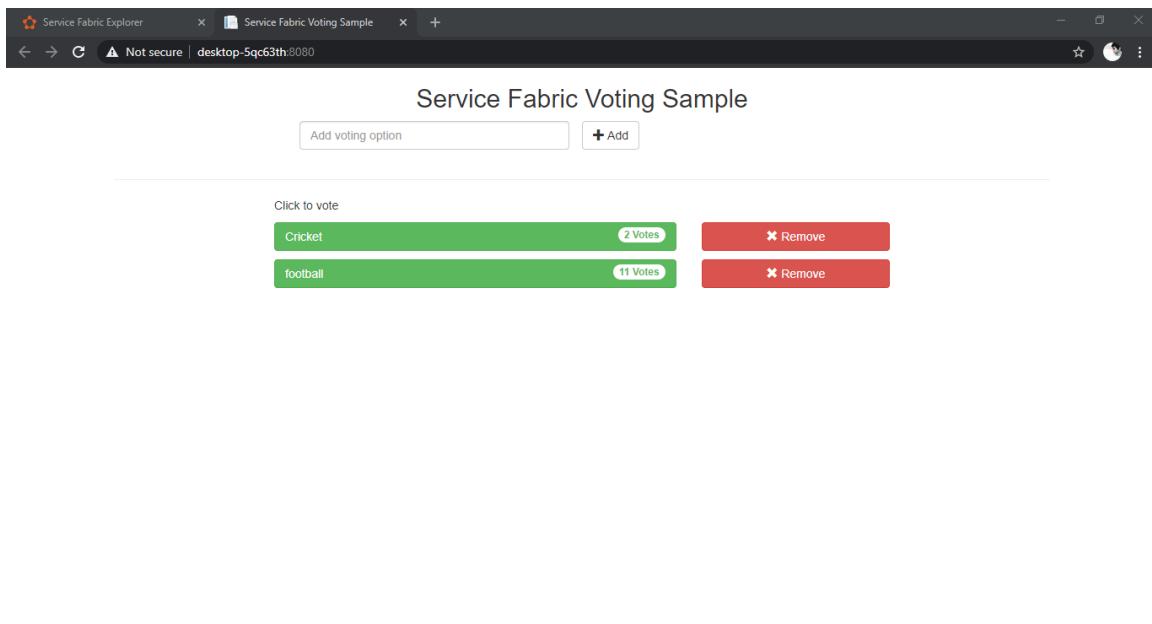


| List of Practical: | |
|---------------------------|--|
| 1. | Develop an ASP.NET Core MVC based Stateless Web App. |
| 2. | Develop a Spring Boot API. |
| 3. | Create an ASP.NET Core Web API and configure monitoring. |
| 4. | a. Create an Azure Kubernetes Service Cluster b. Enable Azure Dev Spaces on an AKS Cluster c. Configure Visual Studio to Work with an Azure Kubernetes Service Cluster d. Configure Visual Studio Code to Work with an Azure Kubernetes Service Cluster |
| | e. Deploy Application on AKS i. Core Web API ii. Node.js API |
| 5. | Create an AKS cluster a. from the portal b. with Azure CLI |
| 6. | Create an Application Gateway Using Ocelot and Securing APIs with Azure AD. |
| 7. | Create a database design for Microservices an application using the database. |
| 8. | a. Create an API management service b. Create an API gateway service |
| 9. | Demonstrate a. Securing APIs with Azure Active Directory. b. Issuing a custom JWT token using a symmetric signing key c. Pre-Authentication in Azure API Management d. AWS API Gateway Authorizer |
| 10. | Create a serverless API using Azure functions |
| 11. | Create an AWS Lambda function |
| 12. | Build AWS Lambda with AWS API gateway |

PRACTICAL 01

Develop an ASP.NET Core MVC based Stateful Web App



What Is Service Fabric?

Azure Service Fabric is a distributed systems platform that allows you to run, manage, and scale applications in a cluster of nodes, using any OS and any cloud.

What is Service Fabric Clusters?

Service Fabric helps you deploy your microservices on a cluster, which is a set of virtual or physical machines that are interconnected through a network. Each machine inside the cluster is called a node. A cluster can consist of thousands of nodes, depending on resource needs of your application. Each node in a cluster has a Windows service called FabricHost.exe, which makes sure that the other two executables (Fabric.exe and FabricGateway.exe) are always running on the cluster nodes.

What is Model-View-Controller (MVC)?

The Model-View-Controller (MVC) architectural pattern separates an app into three main components: Model, View, and Controller. The MVC pattern helps you create apps that are more testable and easier to update.

Models: Classes that represent the data of the app. The model classes use validation logic to enforce business rules for that data. Typically, model objects retrieve and store model state in a database. In this practical, a Movie model retrieves movie data from a database, provides it to the view or updates it. Updated data is written to a database.

Views: Views are the components that display the app's user interface (UI). Generally, this UI displays the model data.

Controllers: Classes that handle browser requests. They retrieve model data and call view templates that return a response. In an MVC app, the view only displays information; the controller handles and responds to user input and interaction. For example, the controller handles route data and query-string values, and passes these values to the model. The model might use these values to query the database.

Step 1: Create Project

Launch Visual Studio as an administrator.

Create a project with File >New >Project.

In the New Project dialog, choose Cloud > Service Fabric Application.

Name the application Voting and click OK.

On the New Service Fabric Service page, choose Stateless ASP.NET Core, name your service VotingWeb, then click OK.

The next page provides a set of ASP.NET Core project templates. For this practical, choose Web

Application (Model-View-Controller), then click OK.

Visual Studio creates an application and a service project and displays them in Solution Explorer.

Step 2: Create and update the files

In this step we will create and will update file in the VotingWeb project

2.1 Update the site.js file

Open wwwroot/js/site.js. Replace its contents with the following JavaScript used by the Home views, then save your changes.

```

var app = angular.module('VotingApp', ['ui.bootstrap']);
app.run(function () { });

app.controller('VotingAppController', ['$rootScope', '$scope', '$http', '$timeout', function ($rootScope, $scope, $http, $timeout) {

    $scope.refresh = function () {
        $http.get('api/Votes?c=' + new Date().getTime())
            .then(function (data, status) {
                $scope.votes = data;
            }, function (data, status) {
                $scope.votes = undefined;
            });
    };

    $scope.remove = function (item) {
        $http.delete('api/Votes/' + item)
            .then(function (data, status) {
                $scope.refresh();
            })
    };

    $scope.add = function (item) {
        var fd = new FormData();
        fd.append('item', item);
        $http.put('api/Votes/' + item, fd, {
            transformRequest: angular.identity,
            headers: { 'Content-Type': undefined }
        })
            .then(function (data, status) {
                $scope.refresh();
                $scope.item = undefined;
            })
    };
}]);

```

2.2 Update the Index.cshtml file

Open Views/Home/Index.cshtml, the view specific to the Home controller. Replace its contents with the following, then save your changes.

```

@{
    ViewData["Title"] = "Service Fabric Voting Sample";
}

<div ng-controller="VotingAppController" ng-init="refresh()">
    <div class="container-fluid">
        <div class="row">
            <div class="col-xs-8 col-xs-offset-2 text-center">
                <h2>Service Fabric Voting Sample</h2>
            </div>
        </div>

        <div class="row">
            <div class="col-xs-8 col-xs-offset-2">
                <form class="col-xs-12 center-block">
                    <div class="col-xs-6 form-group">
                        <input id="txtAdd" type="text" class="form-control" placeholder="Add voting option" ng-model="item"/>
                    </div>
                    <button id="btnAdd" class="btn btn-default" ng-click="add(item)">
                        <span class="glyphicon glyphicon-plus" aria-hidden="true"></span>
                        Add
                    </button>
                </form>
            </div>
        </div>
    </div>

    <hr/>

    <div class="row">
        <div class="col-xs-8 col-xs-offset-2">
            <div class="row">

```

```

<div class="col-xs-4">
    Click to vote
</div>
</div>
<div class="row top-buffer" ng-repeat="vote in votes.data">
    <div class="col-xs-8">
        <button class="btn btn-success text-left btn-block" ng-click="add(vote.key)">
            <span class="pull-left">
                { { vote.key } }
            </span>
            <span class="badge pull-right">
                { { vote.value } } Votes
            </span>
        </button>
    </div>
    <div class="col-xs-4">
        <button class="btn btn-danger pull-right btn-block" ng-click="remove(vote.key)">
            <span class="glyphicon glyphicon-remove" aria-hidden="true"></span>
            Remove
        </button>
    </div>
    </div>
</div>
</div>

```

2.3 Update the _Layout.cshtml file

Open Views/Shared/_Layout.cshtml, the default layout for the ASP.NET app. Replace its contents with the following, then save your changes.

```

<!DOCTYPE html>
<html ng-app="VotingApp" xmlns:ng="http://angularjs.org">
<head>
    <meta charset="utf-8"/>
    <meta name="viewport" content="width=device-width, initial-scale=1.0"/>
    <title>@ViewData["Title"]</title>

    <link href="~/lib/bootstrap/dist/css/bootstrap.min.css" rel="stylesheet"/>
    <link href="~/css/site.css" rel="stylesheet"/>

</head>
<body>
    <div class="container body-content">
        @RenderBody()
    </div>

    <script src="~/lib/jquery/dist/jquery.js"></script>
    <script src="~/lib/bootstrap/dist/js/bootstrap.js"></script>
    <script src="~/lib/angular/angular.js"></script>
    <script src="~/lib/angular-bootstrap/ui-bootstrap-tpls.js"></script>
    <script src="~/js/site.js"></script>

    @RenderSection("Scripts", required: false)
</body>
</html>

```

2.4 Update the VotingWeb.cs file

Open the VotingWeb.cs file, which creates the ASP.NET Core WebHost inside the stateless service using the WebListener web server.

Replace the content with the following code, then save your changes.

```

namespace VotingWeb
{
    using System;
    using System.Collections.Generic;
    using System.Fabric;
    using System.IO;
    using System.Net.Http;
    using Microsoft.AspNetCore.Hosting;
    using Microsoft.Extensions.Logging;
    using Microsoft.Extensions.DependencyInjection;
    using Microsoft.ServiceFabric.Services.Communication.AspNetCore;

```

```

using Microsoft.ServiceFabric.Services.Communication.Runtime;
using Microsoft.ServiceFabric.Services.Runtime;

internal sealed class VotingWeb : StatelessService
{
    public VotingWeb(StatelessServiceContext context)
        : base(context)
    {
    }

    protected override IEnumerable<ServiceInstanceListener> CreateServiceInstanceListeners()
    {
        return new ServiceInstanceListener[]
        {
            new ServiceInstanceListener(
                serviceContext =>
                    new KestrelCommunicationListener(
                        serviceContext,
                        "ServiceEndpoint",
                        (url, listener) =>
                        {
                            ServiceEventSource.Current.ServiceMessage(serviceContext, $"Starting Kestrel on {url}");
                        }
                    )
                .UseKestrel()
                .ConfigureServices(
                    services => services
                        .AddSingleton<HttpClient>(new HttpClient())
                        .AddSingleton<FabricClient>(new FabricClient())
                        .AddSingleton<StatelessServiceContext>(serviceContext)
                )
                .UseContentRoot(Directory.GetCurrentDirectory())
                .UseStartup<Startup>()
                .UseServiceFabricIntegration(listener, ServiceFabricIntegrationOptions.None)
                .UseUrls(url)
                .Build();
            )));
    }

    internal static Uri GetVotingDataServiceName(ServiceContext context)
    {
        return new Uri($"{context.CodePackageActivationContext.ApplicationName}/VotingData");
    }
}
}

```

2.5 Add the VotesController.cs file

- Add a controller, which defines voting actions. Right-click on the Controllers folder, then select Add->New item->Visual C#->ASP.NET Core->Class. Name the file VotesController.cs, then click Add.
- Replace the VotesController.cs file contents with the following, then save your changes this file is modified to read and write voting data from the back-end service.

```

namespace VotingWeb.Controllers
{
    using System;
    using System.Collections.Generic;
    using System.Fabric;
    using System.Fabric.Query;
    using System.Linq;
    using System.Net.Http;
    using System.Net.Http.Headers;
    using System.Text;
    using System.Threading.Tasks;
    using Microsoft.AspNetCore.Mvc;
    using Newtonsoft.Json;

    [Produces("application/json")]
    [Route("api/[controller]")]
    public class VotesController : Controller
    {

```

```

private readonly HttpClient httpClient;
private readonly FabricClient fabricClient;
private readonly string reverseProxyBaseUri;
private readonly StatelessServiceContext serviceContext;

public VotesController(HttpClient httpClient, StatelessServiceContext context, FabricClient
fabricClient)
{
    this.fabricClient = fabricClient;
    this.httpClient = httpClient;
    this.serviceContext = context;
    this.reverseProxyBaseUri = Environment.GetEnvironmentVariable("ReverseProxyBaseUri");
}

// GET: api/Votes
[HttpGet("")]
public async Task<IActionResult> Get()
{
    Uri serviceName = VotingWeb.GetVotingDataServiceName(this.serviceContext);
    Uri proxyAddress = this.GetProxyAddress(serviceName);

    ServicePartitionList partitions = await
this.fabricClient.QueryManager.GetPartitionListAsync(serviceName);

    List<KeyValuePair<string, int>> result = new List<KeyValuePair<string, int>>();

    foreach (Partition partition in partitions)
    {
        string proxyUrl =
 $"{proxyAddress}/api/VoteData?PartitionKey={((Int64RangePartitionInformation)
partition.PartitionInformation).LowKey}&PartitionKind=Int64Range";

        using (HttpResponseMessage response = await this.httpClient.GetAsync(proxyUrl))
        {
            if (response.StatusCode != System.Net.HttpStatusCode.OK)
            {
                continue;
            }

            result.AddRange(JsonConvert.DeserializeObject<List<KeyValuePair<string, int>>>(await
response.Content.ReadAsStringAsync()));
        }
    }

    return this.Json(result);
}

// PUT: api/Votes/name
[HttpPut("{name}")]
public async Task<IActionResult> Put(string name)
{
    Uri serviceName = VotingWeb.GetVotingDataServiceName(this.serviceContext);
    Uri proxyAddress = this.GetProxyAddress(serviceName);
    long partitionKey = this.GetPartitionKey(name);
    string proxyUrl =
 $"{proxyAddress}/api/VoteData/{name}?PartitionKey={partitionKey}&PartitionKind=Int64Range";

    StringContent putContent = new StringContent($"{{ 'name' : '{name}' }}", Encoding.UTF8,
"application/json");
    putContent.Headers.ContentType = new MediaTypeHeaderValue("application/json");

    using (HttpResponseMessage response = await this.httpClient.PutAsync(proxyUrl, putContent))
    {
        return new ContentResult()
        {
            StatusCode = (int) response.StatusCode,
            Content = await response.Content.ReadAsStringAsync()
        };
    }
}

// DELETE: api/Votes/name
[HttpDelete("{name}")]

```

```

public async Task<IActionResult> Delete(string name)
{
    Uri serviceName = VotingWeb.GetVotingDataServiceName(this.serviceContext);
    Uri proxyAddress = this.GetProxyAddress(serviceName);
    long partitionKey = this.GetPartitionKey(name);
    string proxyUrl =
        $"{proxyAddress}/api/VoteData/{name}?PartitionKey={partitionKey}&PartitionKind=Int64Range";

    using (HttpResponseMessage response = await this.httpClient.DeleteAsync(proxyUrl))
    {
        if (response.StatusCode != System.Net.HttpStatusCode.OK)
        {
            return this.StatusCode((int) response.StatusCode);
        }
    }

    return new OkResult();
}

/// <summary>
/// Constructs a reverse proxy URL for a given service.
/// Example: http://localhost:19081/VotingApplication/VotingData/
/// </summary>
/// <param name="serviceName"></param>
/// <returns></returns>
private Uri GetProxyAddress(Uri serviceName)
{
    return new Uri($"{this.reverseProxyBaseUri}{serviceName.AbsolutePath}");
}

/// <summary>
/// Creates a partition key from the given name.
/// Uses the zero-based numeric position in the alphabet of the first letter of the name (0-25).
/// </summary>
/// <param name="name"></param>
/// <returns></returns>
private long GetPartitionKey(string name)
{
    return Char.ToUpper(name.First()) - 'A';
}
}
}

```

Step 3: Add a stateful back-end service to your application

- In Solution Explorer, right-click Services within the Voting application project and choose Add -> New Service Fabric Service....
- In the New Service Fabric Service dialog, choose Stateful ASP.NET Core, name the service VotingData, then press Create.

Once your service project is created, you have two services in your application. As you continue to build your application, you can add more services in the same way. Each can be independently versioned and upgraded.

The next page provides a set of ASP.NET Core project templates. choose API.

Visual Studio creates the VotingData service project and displays it in Solution Explorer.

3.1 Add the VoteDataController.cs file

- In the VotingData project, right-click on the Controllers folder, then select Add->New item->Class. Name the file VoteDataController.cs and click Add.
- Replace the file contents with the following, then save your changes.

```

namespace VotingData.Controllers
{
    using System.Collections.Generic;
    using System.Threading;

```

```

using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using Microsoft.ServiceFabric.Data;
using Microsoft.ServiceFabric.Data.Collections;

[Route("api/[controller]")]
public class VoteDataController : Controller
{
    private readonly IReliableStateManager stateManager;

    public VoteDataController(IReliableStateManager stateManager)
    {
        this.stateManager = stateManager;
    }

    // GET api/VoteData
    [HttpGet]
    public async Task<IActionResult> Get()
    {
        CancellationToken ct = new CancellationToken();

        IReliableDictionary<string, int> votesDictionary =
            await this.stateManager.GetOrAddAsync<IReliableDictionary<string, int>>("counts");

        using (ITransaction tx = this.stateManager.CreateTransaction())
        {
            var list = await votesDictionary.CreateEnumerableAsync(tx);

            var enumerator = list.GetAsyncEnumerator();

            List<KeyValuePair<string, int>> result = new List<KeyValuePair<string, int>>();

            while (await enumerator.MoveNextAsync(ct))
            {
                result.Add(enumerator.Current);
            }
        }

        return this.Json(result);
    }
}

// PUT api/VoteData/name
[HttpPut("{name}")]
public async Task<IActionResult> Put(string name)
{
    IReliableDictionary<string, int> votesDictionary = await
this.stateManager.GetOrAddAsync<IReliableDictionary<string, int>>("counts");

    using (ITransaction tx = this.stateManager.CreateTransaction())
    {
        await votesDictionary.AddOrUpdateAsync(tx, name, 1, (key, oldvalue) => oldvalue + 1);
        await tx.CommitAsync();
    }

    return new OkResult();
}

// DELETE api/VoteData/name
[HttpDelete("{name}")]
public async Task<IActionResult> Delete(string name)
{
    IReliableDictionary<string, int> votesDictionary = await
this.stateManager.GetOrAddAsync<IReliableDictionary<string, int>>("counts");

    using (ITransaction tx = this.stateManager.CreateTransaction())
    {
        if (await votesDictionary.ContainsKeyAsync(tx, name))
        {
            await votesDictionary.TryRemoveAsync(tx, name);
            await tx.CommitAsync();
            return new OkResult();
        }
        else
    }
}

```

```
        {
            return new NotFoundResult();
        }
    }
}
```

Step 4: Publish Service Fabric application

- a. In the solution Explorer Right click on the Voting and select Publish. The Publish dialog box appears.
- b. To open the service fabric explorer, open the Browser and paste the following IP address
<http://localhost:19080/>
- c. Go to cluster >Application >Voting Type >fabric:/Voting >fabric:/VotingWeb > 8482ef73-476d-45a6-aca9-b33d75575855(partition) >_Node_0 Instance
- d. You will see the endpoint in the address section click on that address to run your app

PRACTICAL 2

Develop Spring Boot API

Introduction :-

Spring Boot provides a good platform for Java developers to develop a stand-alone and production-grade spring application that you can just run. You can get started with minimum configurations without the need for an entire Spring configuration setup.

In this example, we showcase that it is possible to host a non-Microsoft stack application to a Service Fabric cluster. We will not create a reliable service; instead, we will host a Java Spring Boot-based API as a guest executable and as a container. We will use VS Code to develop a simple Spring Boot application.

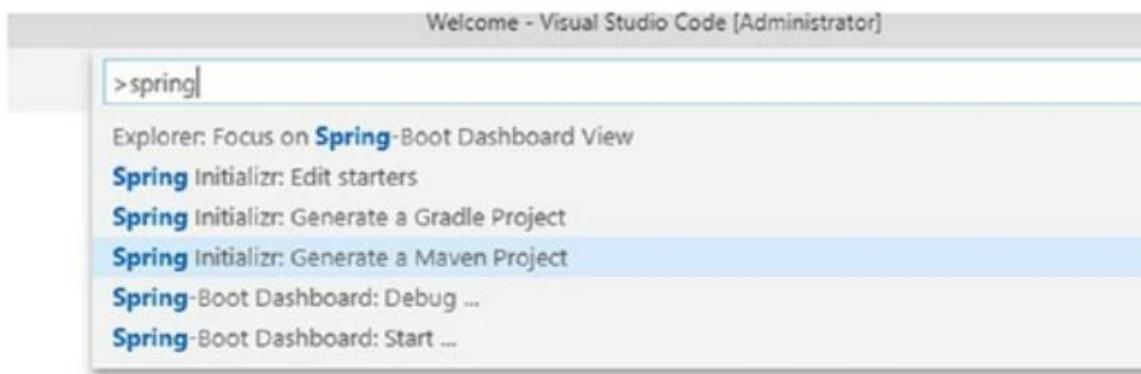
Setting up the Development Environment

Let's set up the development environment.

1. Install Visual Studio 2017.
2. Install the Microsoft Azure Service Fabric SDK.
3. Install Visual Studio Code.
 - a. Install Spring Boot Extensions Pack.
 - b. Install Java Extensions Pack.
 - c. Install Maven for Java.
4. Make sure that the Service Fabric Local cluster is in a running state.
5. Install Docker Desktop.
6. Access the Azure container registry.

Develop a Spring Boot API

Now it's time to get started on the



application.

1. Launch Visual Studio Code as an administrator.
2. Press Ctrl+Shift+P to open the command palette.
3. Enter spring in the command palette, and choose Spring Initializr: Generate a maven project
4. Choose Java for Specify Project Language.
5. Enter com.microservices in the input group ID for your project.
6. Enter employeespringsservice in the input artifact ID for your project.
7. Choose the latest Spring boot version.

8. Choose the following dependencies.

- a. DevTools
- b. Lombok
- c. Web
- d. Actuator

9. Choose the path where you want to save the solution.

10. Right-click the EMPLOYEESSPRINGSERVICE folder under src ► main ► java ► com ► microservices, as shown in Figure 3-15, and click Add File.

Add a new file

11. Name the file Employee.Java and add the following code. (This is the definition of the employee object; we kept it simple by having only three properties to represent an employee.)

```
package com.microservices.employeespringservice;
```

```
import lombok.AllArgsConstructor;
```

```
import lombok.EqualsAndHashCode;
```

```
import lombok.Getter;
```

```
import lombok.Setter;
```

```
/** Employee */
```

```
@Getter
```

```
@Setter
```

```
@EqualsAndHashCode
```

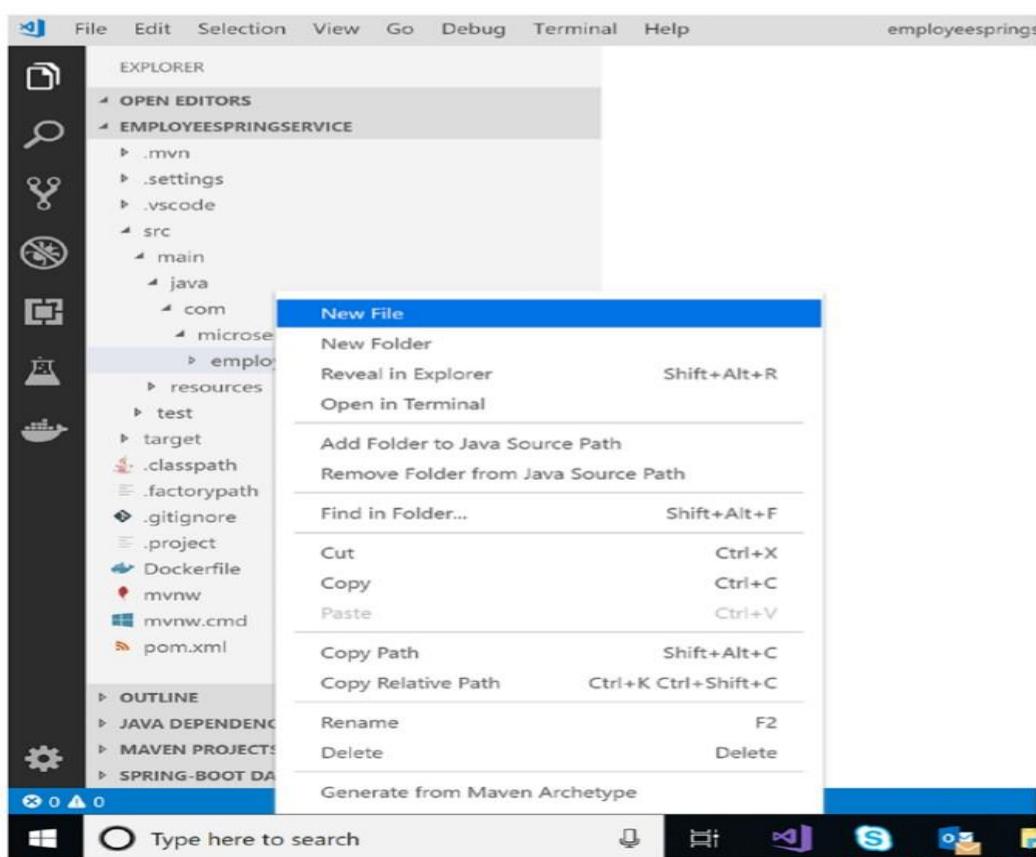
```
@AllArgsConstructor
```

```
public class Employee {
```

```
    private String firstName;
```

```
    private String lastName;
```

```
    private String ipAddress; }
```



12. Now let's create an employee service that returns an

employee's information. Right-click the employeespringservice folder and add a file named EmployeeService.java. Add the following code to it.

```
package com.microservices.employeespringservice;
```

```
import java.net.InetAddress;
```

```
import java.net.UnknownHostException;
```

```
import org.springframework.stereotype.Service;
```

```
/*
```

```
* EmployeeService
```

```

*/
@Service
public class EmployeeService {
    public Employee GetEmployee(String firstName, String
lastName){
        String ipAddress;
        try {
            getHostAddress().toString();
        } catch (UnknownHostException e) {
            ipAddress = e.getMessage();
        }
        Employee employee = new Employee(firstName,
lastName,ipAddress);
        return employee;
    }
}

```

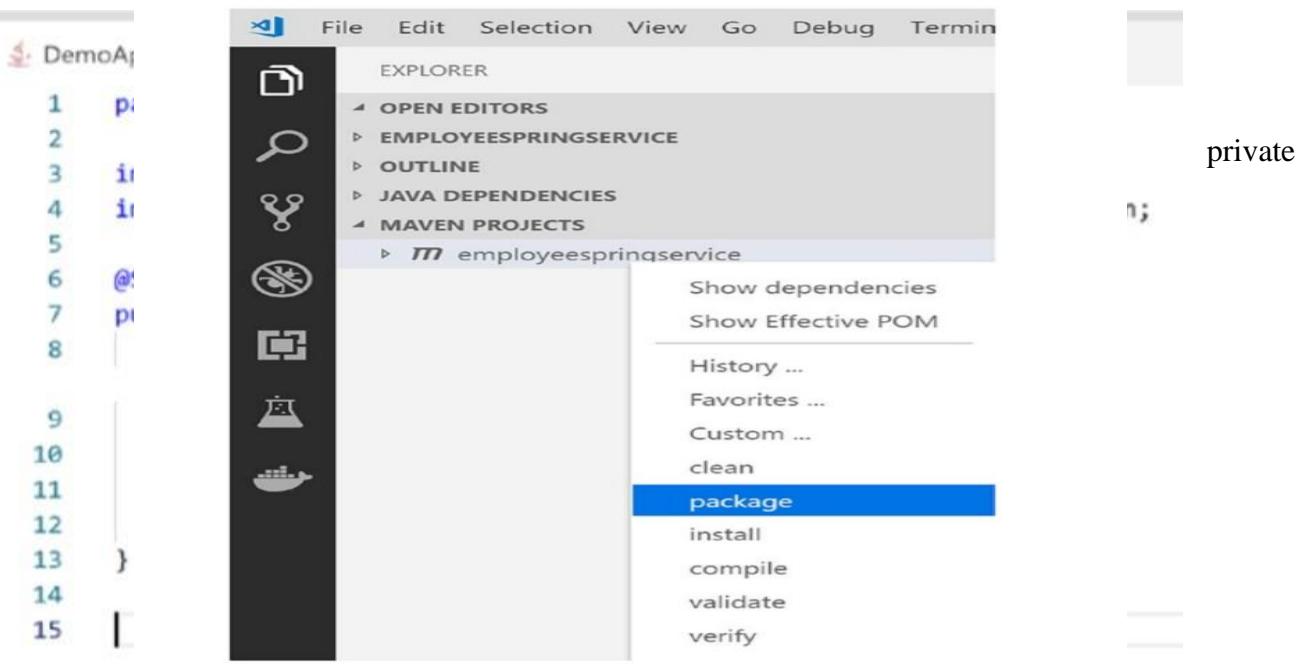
13. Now let's create an employee controller that invokes the employee service to return the details of an employee. Right-click the employeespringservice folder and add a file named EmployeeController.

```

package com.microservices.employeespringservice;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Controller;
import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.ResponseBody;

/**
 * EmployeeController
 */
@Controller
public class EmployeeController {
    @Autowired

```



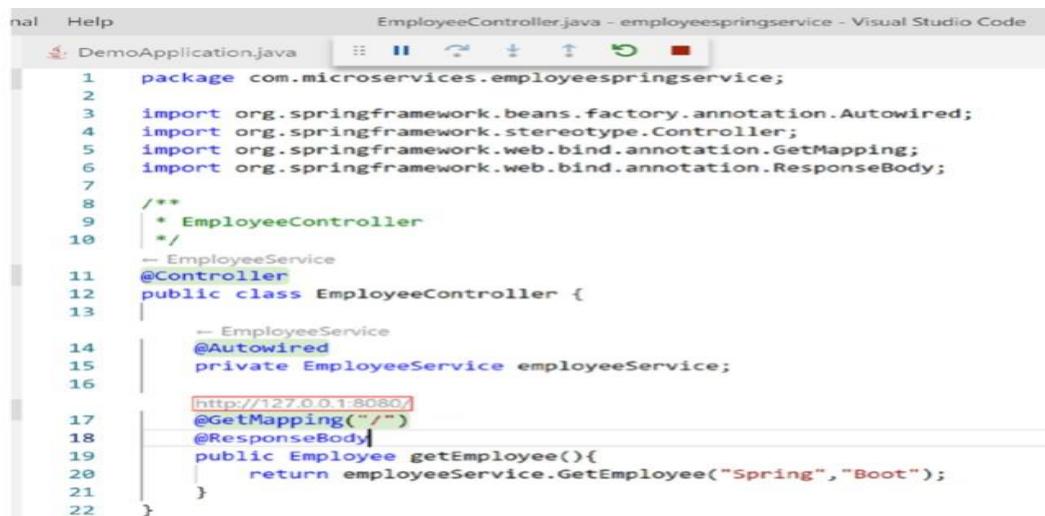
```
EmployeeService employeeService;
@GetMapping("/")
@ResponseBody
public Employee getEmployee(){
    return employeeService.GetEmployee("Spring","Boot");
}
}
```

Now you are ready for a simple REST-based service that returns employee information. Visual Studio Code has some cool features to run Spring Boot applications, and we are going to use the same.

1. Open DemoApplication.java and once you open the file, you see the option to run or debug the application. Please note that this may take time as Visual Studio automatically downloads the dependencies.

2. Click Run and open the Controller class. You see the URL where your controller service is hosted.

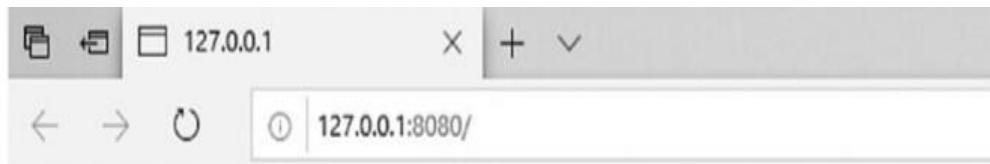
3. Click the URL and you see the output show in your default browser.



```
EmployeeController.java - employeespringservice - Visual Studio Code
1 package com.microservices.employeespringservice;
2
3 import org.springframework.beans.factory.annotation.Autowired;
4 import org.springframework.stereotype.Controller;
5 import org.springframework.web.bind.annotation.GetMapping;
6 import org.springframework.web.bind.annotation.ResponseBody;
7
8 /**
9  * EmployeeController
10 */
11 -- EmployeeService
12 @Controller
13 public class EmployeeController {
14
15     -- EmployeeService
16     @Autowired
17     private EmployeeService employeeService;
18
19     @GetMapping("/")
20     @ResponseBody
21     public Employee getEmployee(){
22         return employeeService.GetEmployee("Spring","Boot");
23     }
24 }
```

4. Right-click

employeespringservice under Maven Projects. Click package, This generates the JAR file.

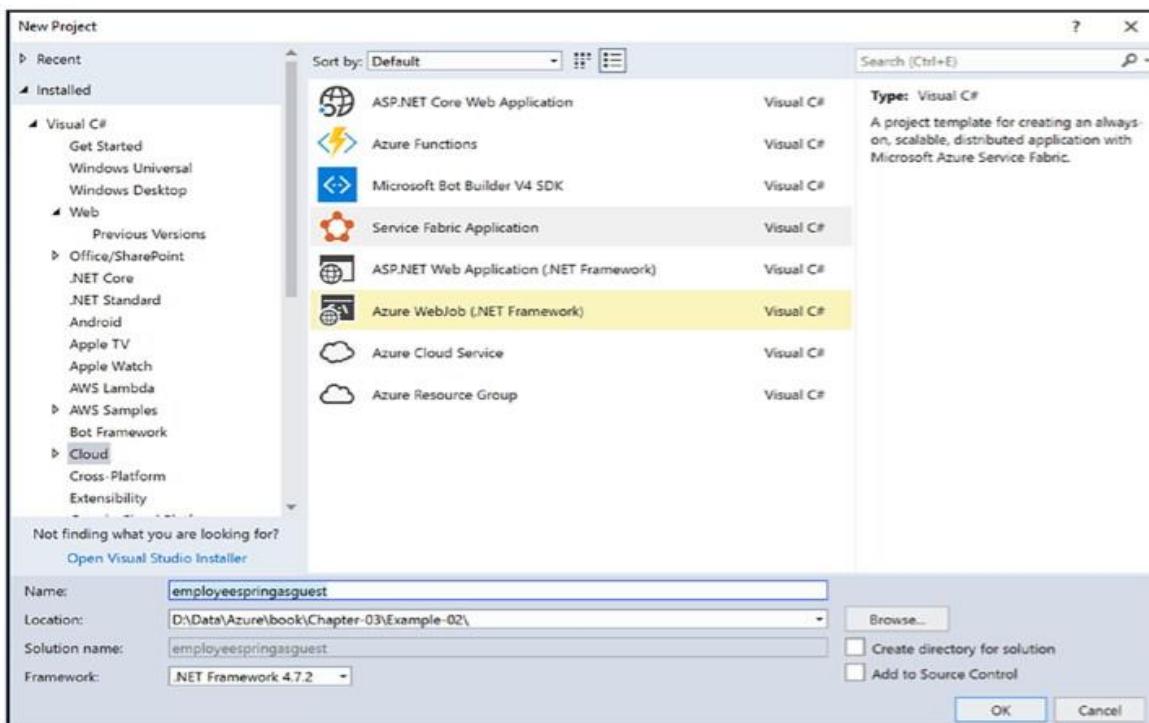


Now you have a simple Spring Boot-based REST API, and you have

generated a JAR file. We will now deploy it to Service Fabric as a guest executable. To deploy, we will use Visual Studio 2017.

Deploy a Spring Boot Service as a Guest Executable

After executing all the steps in the previous section, your development is complete. Please follow

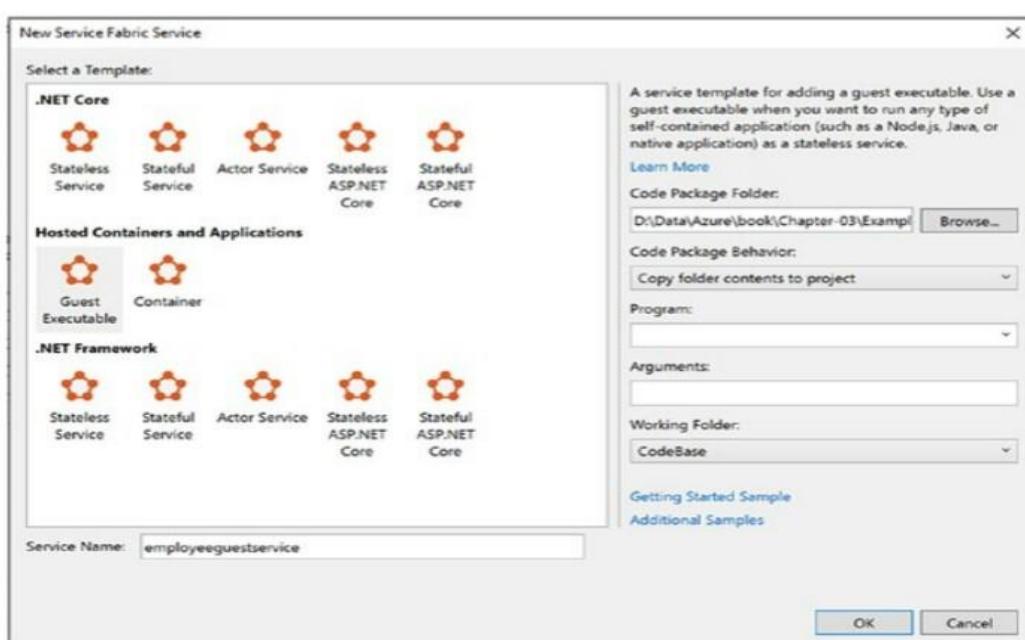


the steps in
this section to

deploy the developed Spring Boot application as a guest executable. This shows that it is possible to host a non-Microsoft stack application on a Service Fabric cluster by using a guest executable programming model. Service Fabric considers guest executables a stateless service.

1. Launch Visual Studio 2017 as an administrator.

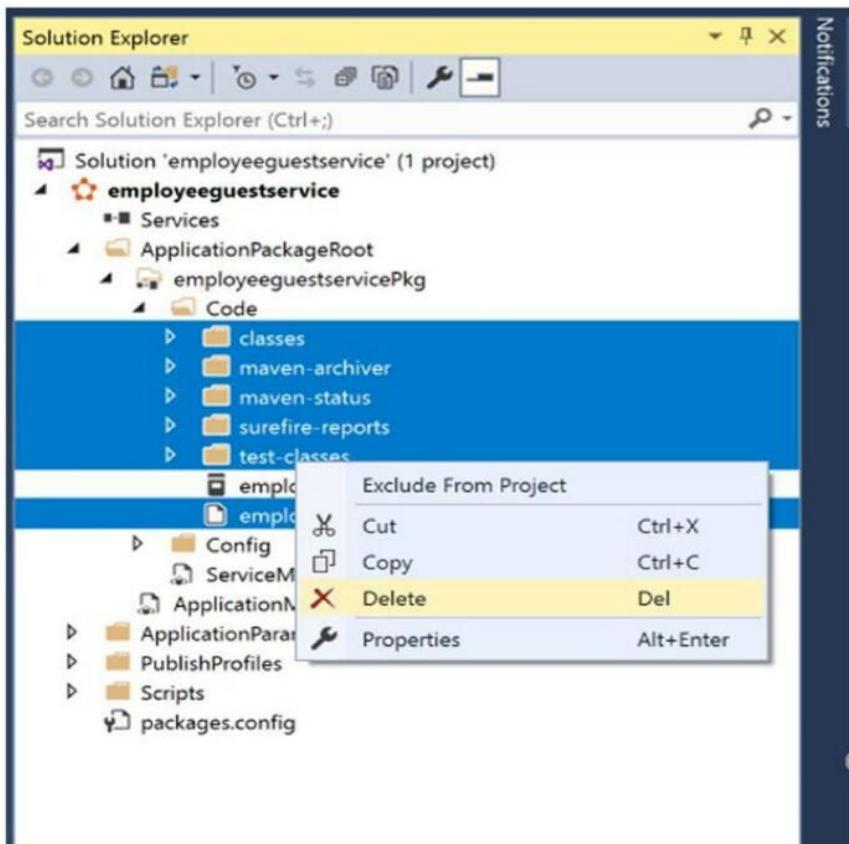
1. Click File ► New Project ► Select Cloud ► Service Fabric Application.
2. Name the application employeespringasguest



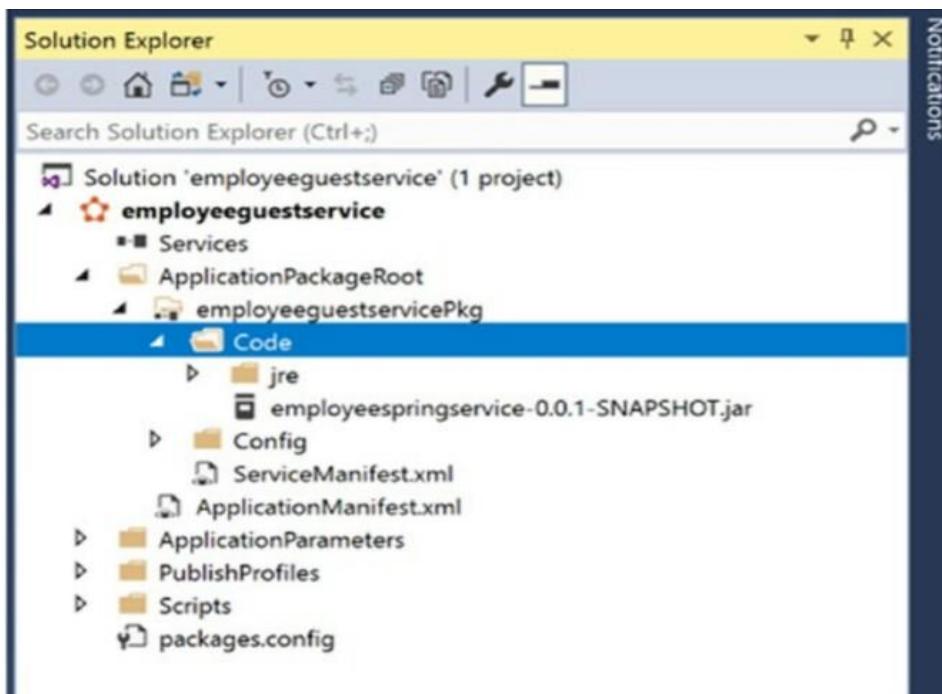
3. In New Service Fabric Service, select the following

- a. Service Name: employeeguestservice
- b. Code Package Folder: Point to the target folder in which Visual Studio Code generated the JAR file for the Spring Boot service.
- c. Code Package Behavior: Copy folder contents to folder
- d. Working Folder: CodeBase

4. Delete the selected files from the Code folder.



5. We also need to upload the runtime to run the JAR. Generally, it resides in the JDK installation folder (C:\java-1.8.0-openjdk-1.8.0.191-1.b12.redhat.windows.x86_64). Paste it in the Code folder.



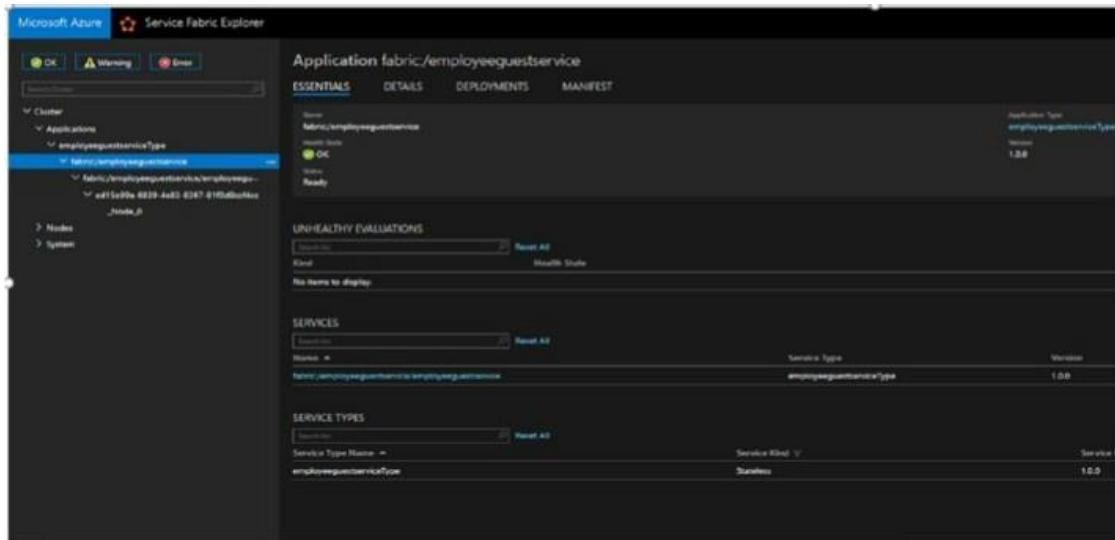
6. Open ServiceManifest.xml and set the following values.

```

<EntryPoint>
<ExeHost>
<Program>jre\bin\java.exe</Program>
<Arguments>-jar ..\..\employeespringservice-0.0.1-
SNAPSHOT.jar</Arguments>
<WorkingFolder>CodeBase</WorkingFolder>
<!-- Uncomment to log console output (both stdout and stderr) to one of the
service's working directories. -->
<!-- <ConsoleRedirection FileRetentionCount="5"
FileMaxSizeInKb="2048"/> -->
</ExeHost>
</EntryPoint>
</CodePackage>
<Resources>
```

<Endpoints>

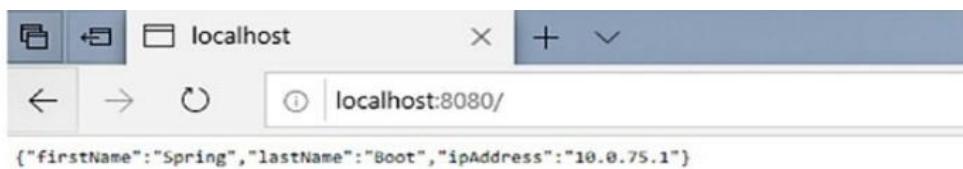
<!-- This endpoint is used by the communication listener to obtain the port on which to listen. Please note that if your service is partitioned, this port is shared with replicas of different partitions that are placed in



<Endpoint Name="employeeguestserviceTypeEndpoint"

Protocol="http" Port="8080" Type="Input" />

</Endpoints>



</Resources>

7. Make sure that the local Service Fabric cluster is up and running. Click F5. Browse the Service Fabric dashboard. The default URL is <http://localhost:19080/Explorer/index.html>. You see that your service is deployed.

8. Browse <http://localhost:8080> to access your service. In servicemanifest.xml, we specified the service port as 8080; you can browse the same on 8080.

Deploy a Spring Boot Service as a Container

So far, we have deployed the service as a guest executable in Service Fabric. Now we will follow the steps to deploy the Spring service as a container in Service Fabric. This explains that in addition to creating stateful and stateless services, Service Fabric also orchestrates containers like any other orchestrator, even if the application wasn't developed on a Microsoft stack.

1. Open Visual Studio Code. Open the folder where employeespringservice exists. Open the Docker file.
2. Make sure that the name of the JAR file is correct.
3. Select Switch to Windows container... in Docker Desktop.



4. Create the Azure Container Registry resource in the Azure portal. Enable the admin user.

A screenshot of the Azure portal showing the 'myservicefabric - Access keys' blade. On the left, there is a navigation sidebar with options like Overview, Activity log, Access control (IAM), Tags, Quick start, Events, Settings, Access keys (which is selected and highlighted in blue), Locks, Automation script, Services, Repositories, Webhooks, Replications, Policies, Content trust (Preview), Monitoring, Metrics (Preview), Support + troubleshooting, and New support request. The main content area shows the 'Registry name' as 'myservicefabric' and the 'Login server' as 'myservicefabric.azurecr.io'. Under 'Access keys', there is a section for 'Admin user' with 'Enable' and 'Disable' buttons, both of which are currently 'Enable'. Below this, there is a 'Username' field containing 'myservicefabric' and a table for 'NAME' and 'PASSWORD' with two rows: 'password' and 'password2'. Both password fields are redacted with red bars.

5. Open the command prompt in Administrative Mode and browse to the directory where the Docker file exists.
6. Fire the following command, including the period at the end. (This may take time because it downloads the Window Server core image from the Docker hub)
`docker build -t employeespringservice/v1 .`

```
Administrator: Command Prompt
D:\Data\Azure\book\Chapter-03\Example-02\employeespringservice>docker build -t employeespringservice/v1 .
Sending build context to Docker daemon 20.06MB
Step 1/4 : FROM openjdk:windowsservercore
--> b76400672bb0
Step 2/4 : EXPOSE 8080
--> Using cache
--> e6ad19873371
Step 3/4 : ADD /target/employeespringservice-0.0.1-SNAPSHOT.jar employeespringservice-0.0.1-SNAPSHOT.jar
--> 72fa3bfa7e64
Step 4/4 : ENTRYPOINT ["java","-jar","employeespringservice-0.0.1-SNAPSHOT.jar"]
--> Running in 9a87f89f2736
Removing intermediate container 9a87f89f2736
--> 9ab1c34d61f7
Successfully built 9ab1c34d61f7
Successfully tagged employeespringservice/v1:latest

D:\Data\Azure\book\Chapter-03\Example-02\employeespringservice>docker login [REDACTED].azurecr.io -u [REDACTED]
WARNING! Using --password via the CLI is insecure. Use --password-stdin.
Login Succeeded

D:\Data\Azure\book\Chapter-03\Example-02\employeespringservice>
```

Now the container image is available locally. You have to push the image to Azure Container

```
Administrator: Command Prompt
D:\Data\Azure\book\Chapter-03\Example-02\employeespringservice>docker build -t employeespringservice/v1 .
Sending build context to Docker daemon 20.06MB
Step 1/4 : FROM openjdk:windowsservercore
--> b76400672bb0
Step 2/4 : EXPOSE 8080
--> Using cache
--> e6ad19873371
Step 3/4 : ADD /target/employeespringservice-0.0.1-SNAPSHOT.jar employeespringservice-0.0.1-SNAPSHOT.jar
--> 72fa3bfa7e64
Step 4/4 : ENTRYPOINT ["java","-jar","employeespringservice-0.0.1-SNAPSHOT.jar"]
--> Running in 9a87f89f2736
Removing intermediate container 9a87f89f2736
--> 9ab1c34d61f7
Successfully built 9ab1c34d61f7
Successfully tagged employeespringservice/v1:latest

D:\Data\Azure\book\Chapter-03\Example-02\employeespringservice>
```

Registry.

1. Log in to Azure Container Registry using the admin username and password. Use the following command .

```
docker login youracr.azurecr.io -u yourusername -p yourpassword
```

2. Fire the following commands to upload the image to ACR.

```
docker tag employeespringservice/v1 youracr.azurecr.io/book/
```

```
Administrator: Command Prompt
employeeservice/v1          latest      154a12b32578   3 weeks ago  4.52 ▾
GB
myservicefabric.azurecr.io/samples/employeeservice/v1  latest      154a12b32578   3 weeks ago  4.52
GB
openjdk                   windowsservercore  b76400672bb0   3 weeks ago  4.50
B

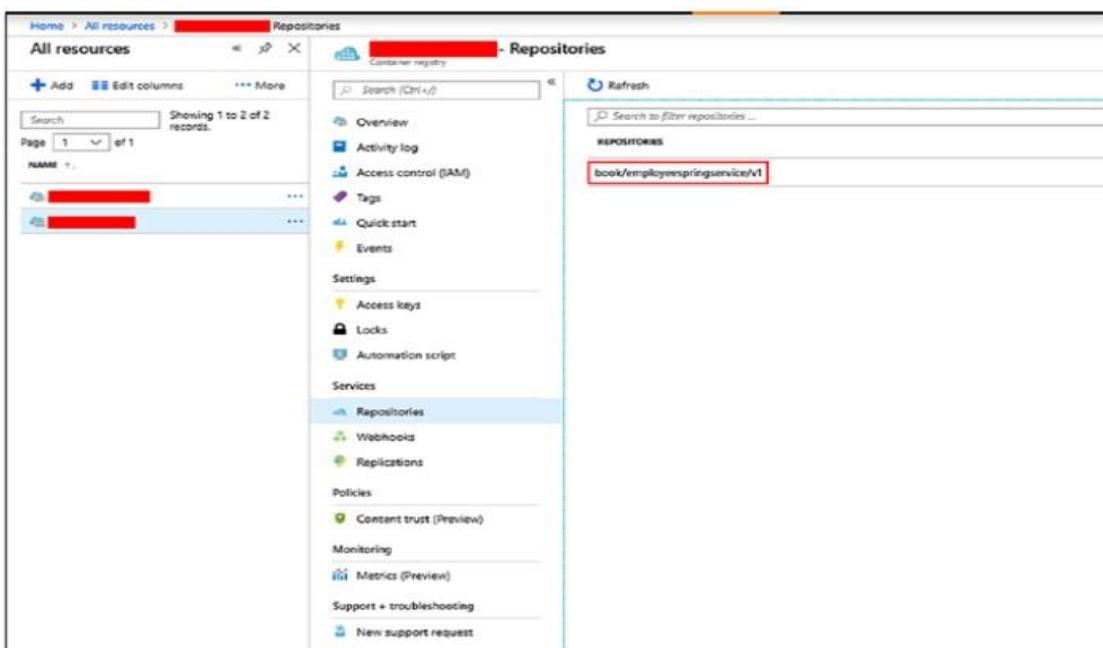
D:\Data\Azure\book\Chapter-03\Example-02\employeespringservice>docker tag employeespringservice/v1 myservicefabric.azure
cr.io/book/employeespringservice/v1

D:\Data\Azure\book\Chapter-03\Example-02\employeespringservice>docker push [REDACTED].azurecr.io/book/employeesprin
gservice/v1
The push refers to repository [REDACTED].azurecr.io/book/employeespringservice/v1]
Bba5f91bb947: Pushed
c309774bb359: Pushed
a07ccaa47fd47: Pushed
aff8ea0481d2: Pushed
664240f6bb32: Pushed
f9da363fd495: Pushed
6ed33c44b167: Pushed
98331e8a4501: Pushed
176c7b727a0b: Pushed
e388c396e652: Pushed
f0f0b327dc34: Pushed
```

```
employeespringservice/v1  
docker push myservicefabric.azurecr.io/book/  
employeespringservice/v1
```

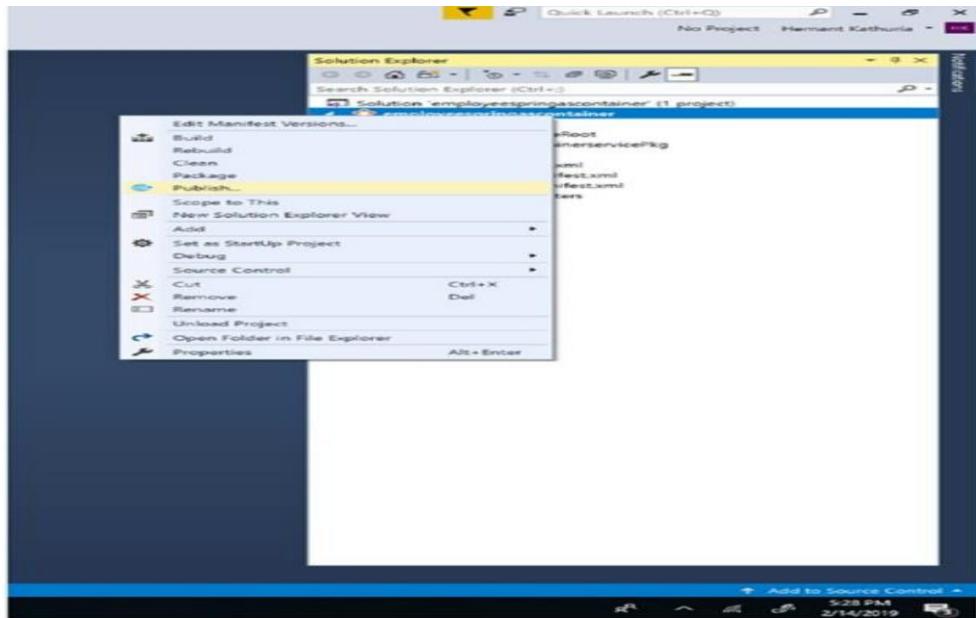
3. Log in to the Azure portal and check if you can see your image in Repositories. Since the container image is ready and uploaded in Azure Container Registry, let's create a Service Fabric project to deploy the container to the local Service Fabric cluster.

1. Launch Visual Studio 2017 as an administrator.
2. Click File ► New Project ► Select Cloud ► Service Fabric Application.
3. Name the application employeespringascontainer.



4. In New Service Fabric Service, select the following.
 - a. Service Name: employeecontainerservice
 - b. Image Name: youracr.azurecr.io/book/employeespringservice/v1

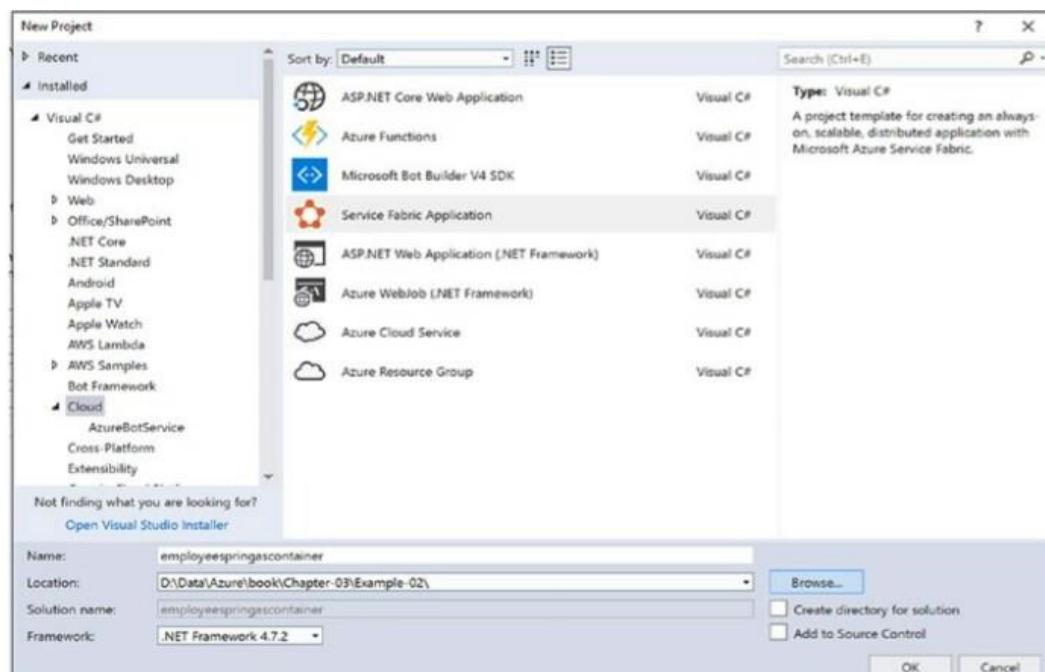
- c. User Name: Your username in the Azure Container Registry
 - d. Host Port: 8090
 - e. Container Port: 8080
5. Once the solution is created, open the ApplicationManifest.xml. Specify the right password for the admin user. (Since this is a sample, we kept the password unencrypted; for real-word applications you have to encrypt the password)



Now we are ready to build and deploy the

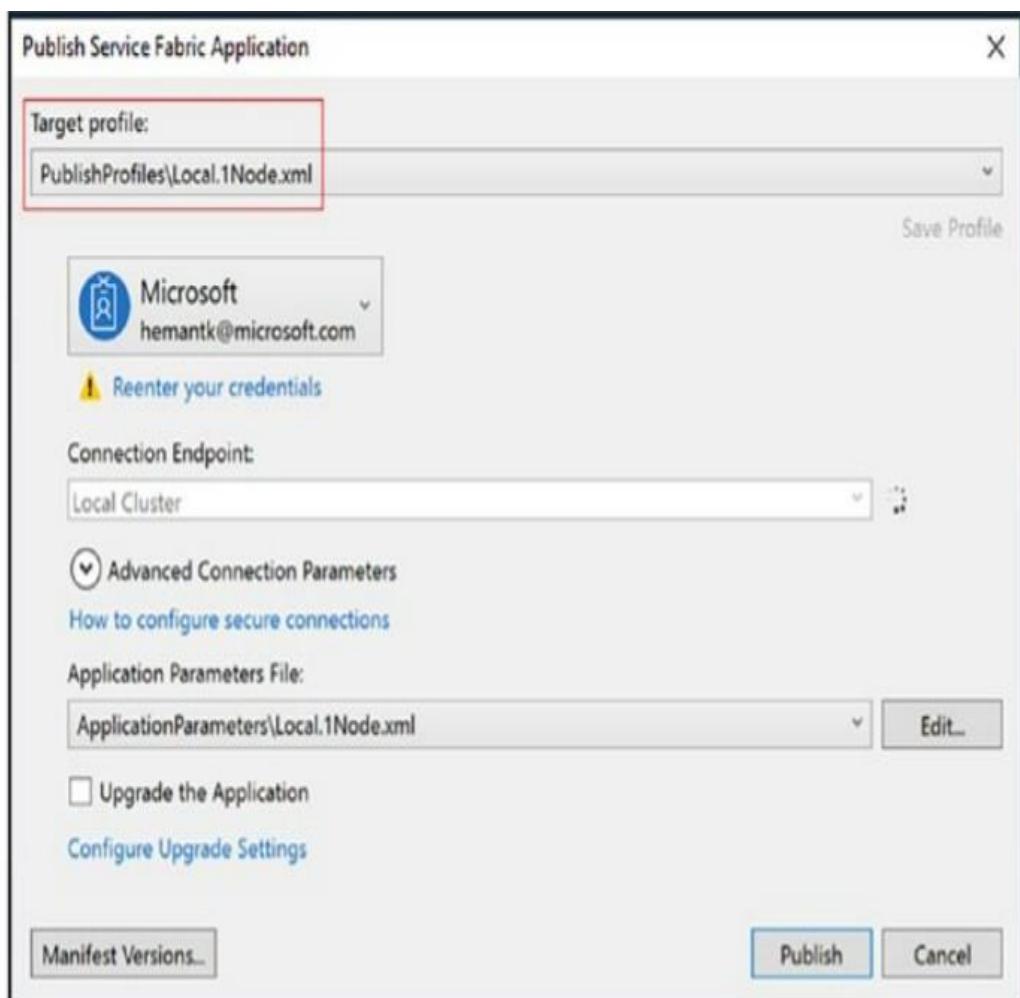
container to the local Service Fabric cluster. Since we have given the user information to download the image from Azure Container Registry, Visual Studio downloads and deploy the container to the local services fabric cluster.

1. Right-click the Service Fabric project and publish.



2. Select the local cluster profile to publish to the local Service Fabric cluster. To deploy to Azure, select the cloud profile. Make sure that the

Service Fabric cluster is up and ready in your subscription



3. Browse the Service Fabric dashboard. The default URL is <http://localhost:19080/Explorer/index.html>. Your service is deployed.

```
applicationManifest.xml <?xml version="1.0" encoding="utf-8"?>
<Fabric xmlns="http://schemas.microsoft.com/2011/01/fabric"
        xmlns:xsd="http://www.w3.org/2001/XMLSchema"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
    <Parameters>
        <Parameter Name="employeecontainerservice_InstanceCount" DefaultValue="-1" />
    </Parameters>
    <Imports>
        <Import Uri="EmployeeContainerServiceManifest.xml" />
    </Imports>
    <ServiceManifestImports>
        <ServiceManifestImport ServiceManifestName="EmployeeContainerServicePkg" ServiceManifestType="Application" />
    </ServiceManifestImports>
    <ConfigOverrides />
    <Policies>
        <ContainerHostPolicies CodePackageRef="Code">
            <!-- See https://aka.ms/I7zQn9 for how to encrypt your repository password -->
            <PortBinding ContainerPort="8080" EndpointRef="employeecontainerservice">
                <ContainerPortRef />
            </PortBinding>
        </ContainerHostPolicies>
    </Policies>
    <ServiceManifestImports>
        <ServiceManifestImport ServiceManifestName="EmployeeContainerServicePkg" ServiceManifestType="Application" />
    </ServiceManifestImports>
    <DefaultServices>
        <!-- The application below creates instances of service types, when an instance of the application type is created. You can also create one or more instances of the service type using the ServiceFabric PowerShell module. -->
        <Service Name="EmployeeContainerService" ServiceTypeRef="EmployeeContainerServiceType" InstanceCount="1" />
        <StatelessService ServiceName="EmployeeContainerServiceType" InstanceCount="1" />
        <ServicePartition />
        <Service>
            <StatelessService />
        </Service>
    </DefaultServices>
    <ApplicationManifest />
</Fabric>
```

Microsoft Azure Service Fabric Explorer

Application fabric/employeespringascontainer

ESSENTIALS DETAILS DEPLOYMENTS MANIFEST

Cluster Applications employeeSpringContainerType employeeSpringContainerType

> Microsoft/employeespringascontainer

Health State: OK Status: Ready

UNHEALTHY EVALUATIONS

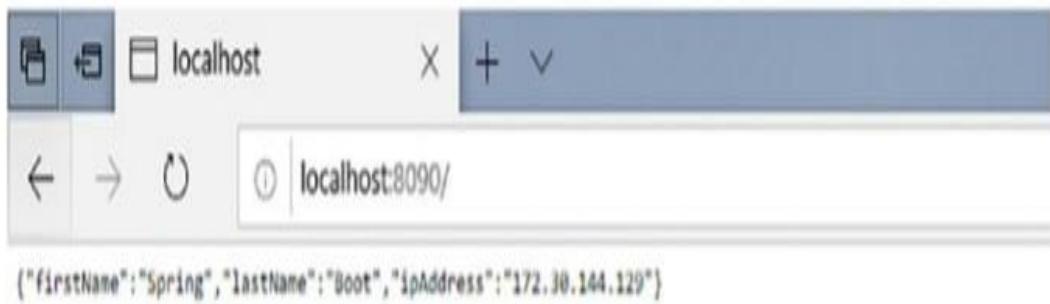
SERVICES

| Name | Service Type | Version |
|--|------------------------------|---------|
| fabric:/EmployeeSpringContainerType/EmployeeContainerService | employeeContainerServiceType | 1.0.0 |

SERVICE TYPES

| Service Type Name | Service End V | Service |
|------------------------------|---------------|----------|
| employeeContainerServiceType | 0.0.0 | Services |

4. Browse to <http://localhost:8090/> to access your service. You get the response which is served from the container run by Service Fabric.



PRACTICAL NO:3

Create an Asp.Net Web Core Api and Configure Monitoring.

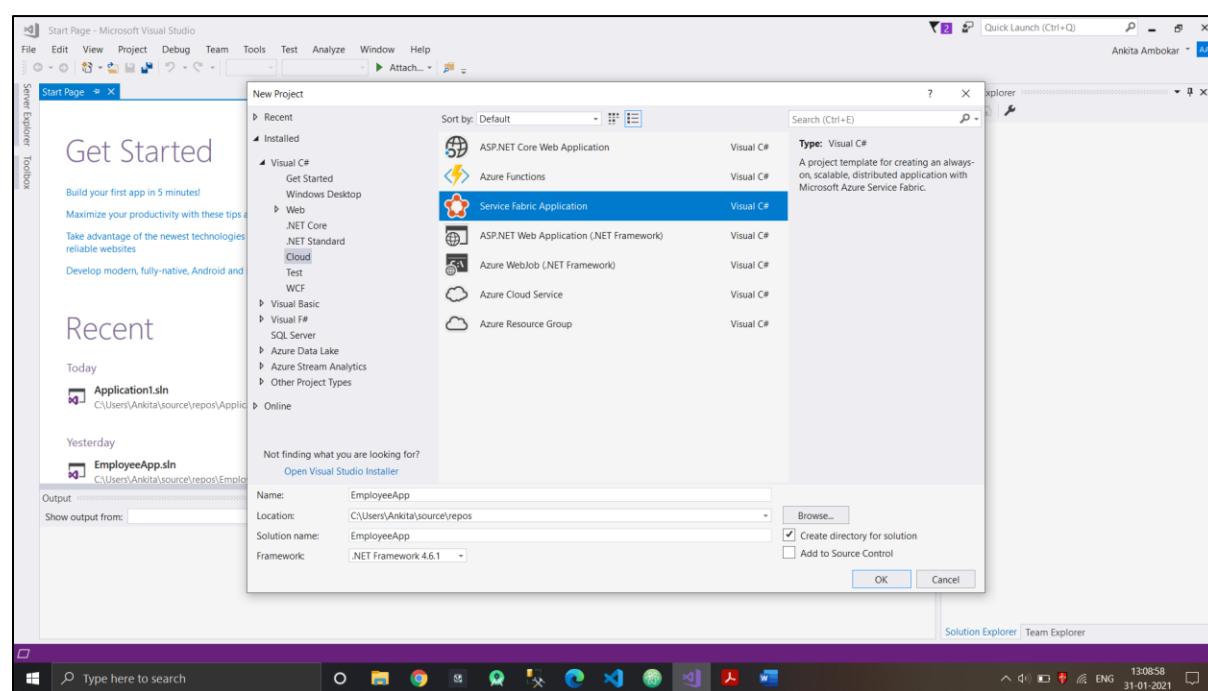
Setting up the Development Environment
Let's set up.

1. Install Visual Studio 2017.
2. Install the Microsoft Azure Service Fabric SDK.
3. Create the Translator Text API in your Azure subscription and make a note of the access key.
4. Create an empty Azure SQL Database and keep the connection string with SQL Authentication handy.
5. Make sure that the Service Fabric local cluster on Windows is in a running state.
6. Make sure that the Service Fabric Azure cluster on Windows is in a running state.

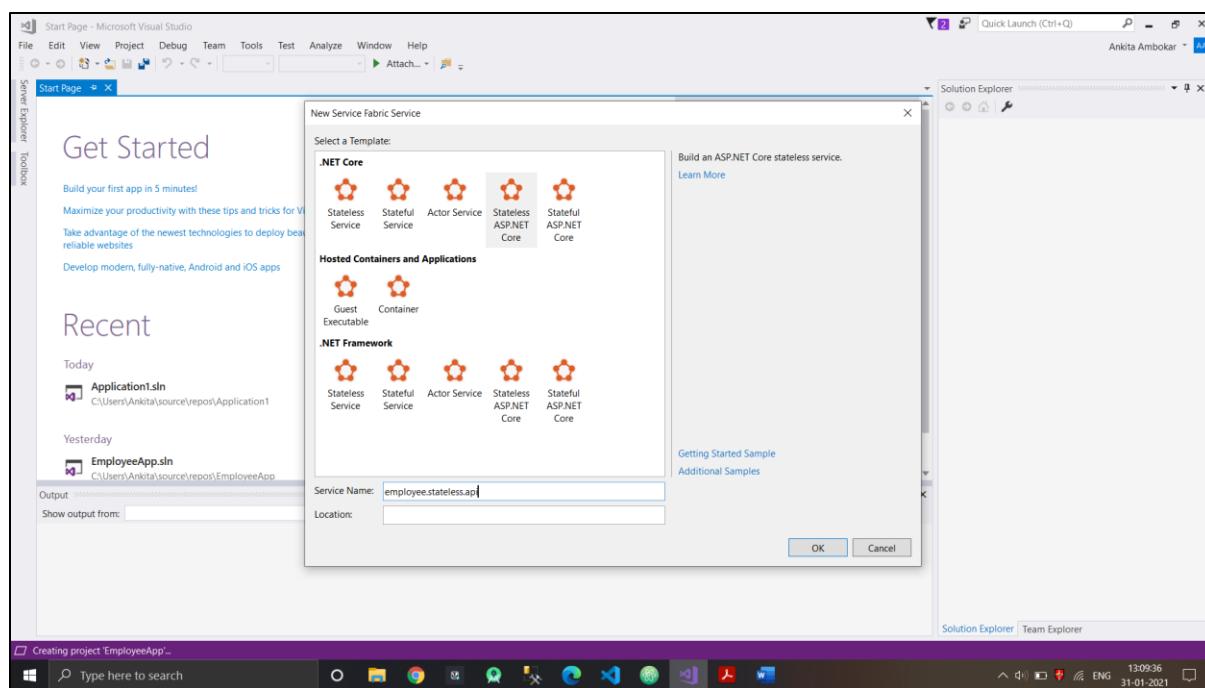
Create an ASP.NET Core Web API

Now let's start the API.

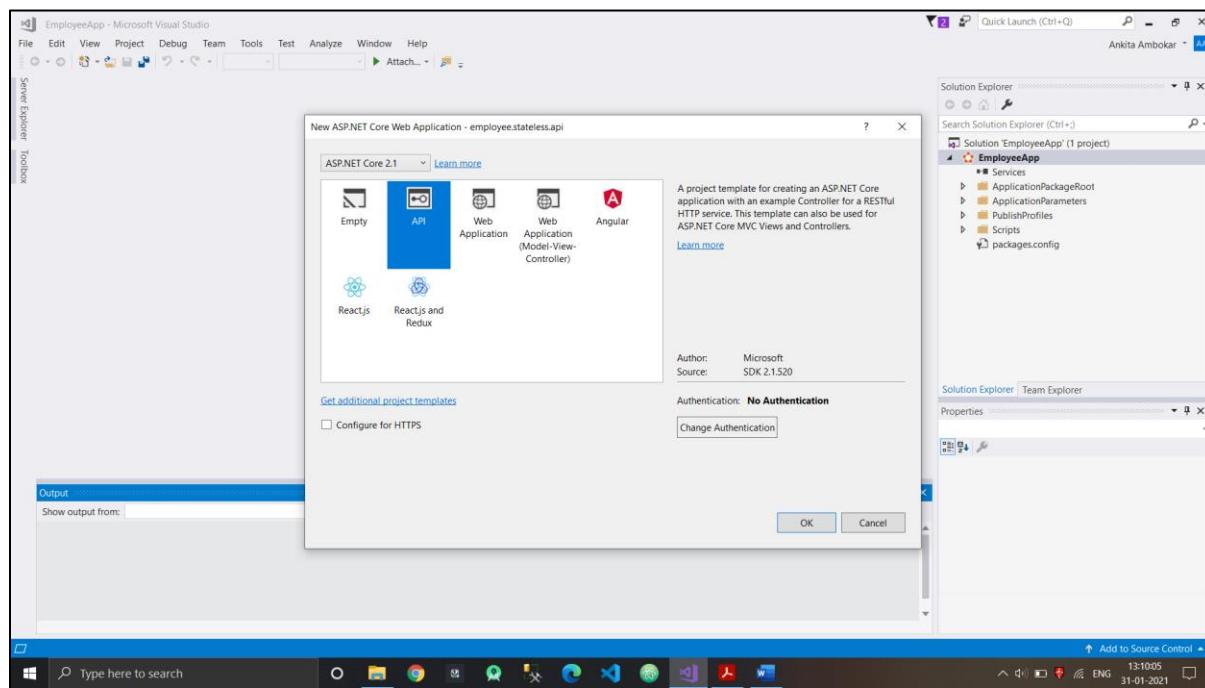
1. Launch Visual Studio 2017 as an administrator.
2. Create a project by selecting File → New → Project.
3. In the New Project dialog, choose Cloud → Service Fabric Application.
4. Name the Service Fabric application **EmployeeApp** and click OK.



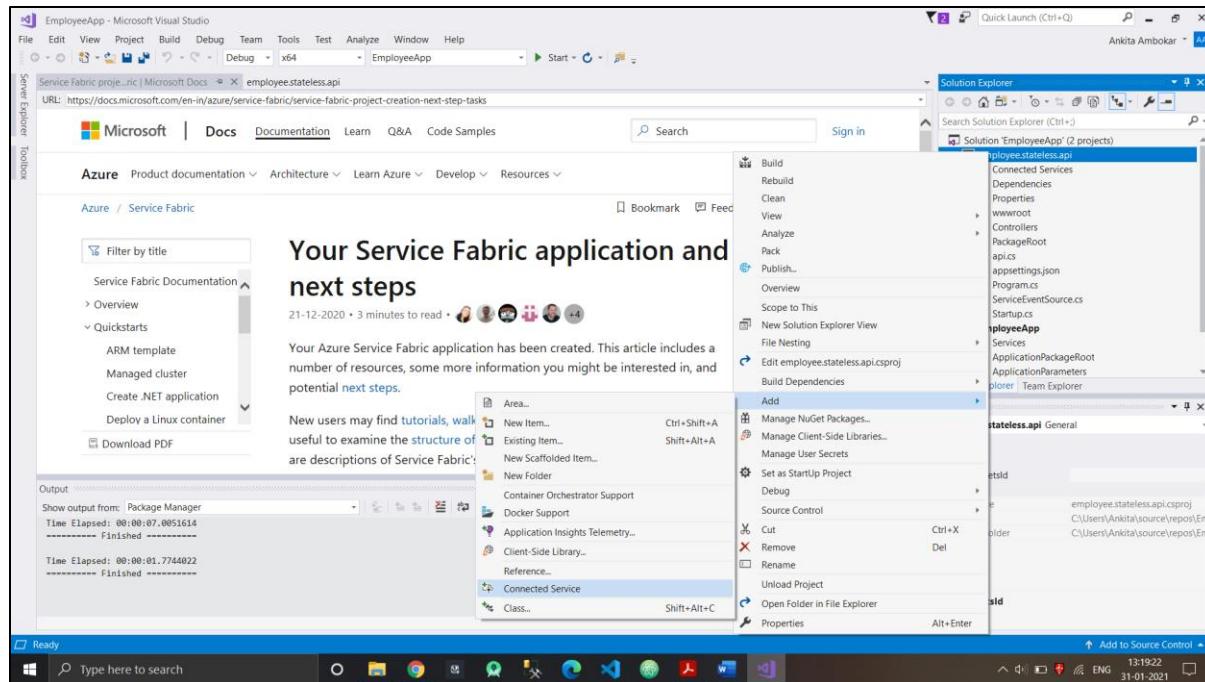
5. Name the stateless ASP.NET Core service **Employee.Stateless.Api** (as seen in Figure) and click OK.



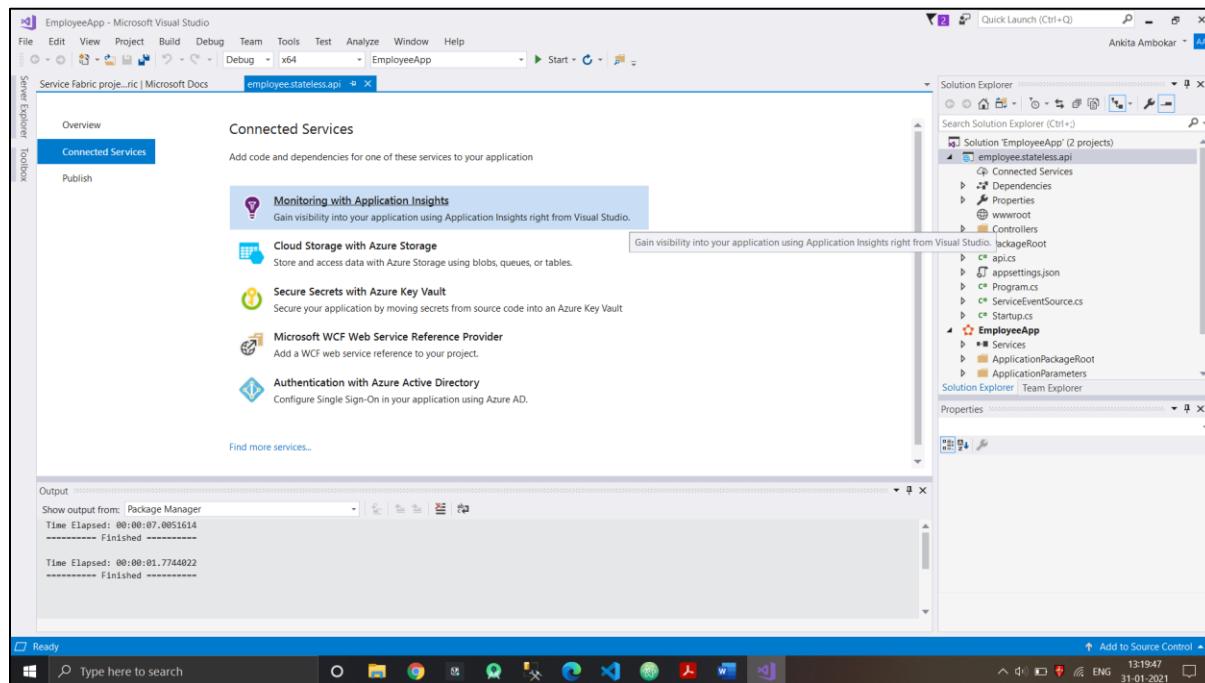
6. Choose the API and click OK. Make sure that ASP.NET Core 2.2 is selected, as shown in Figure.



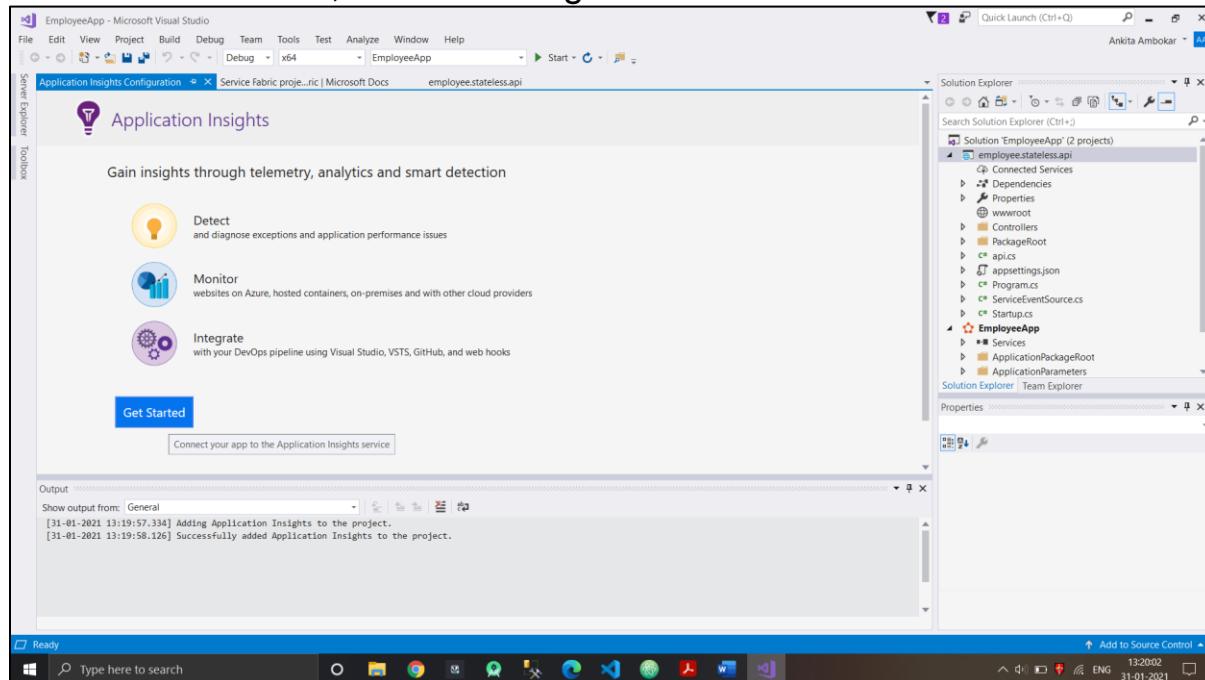
7. Right-click the employee.stateless.api project and select Add → Connected Service.



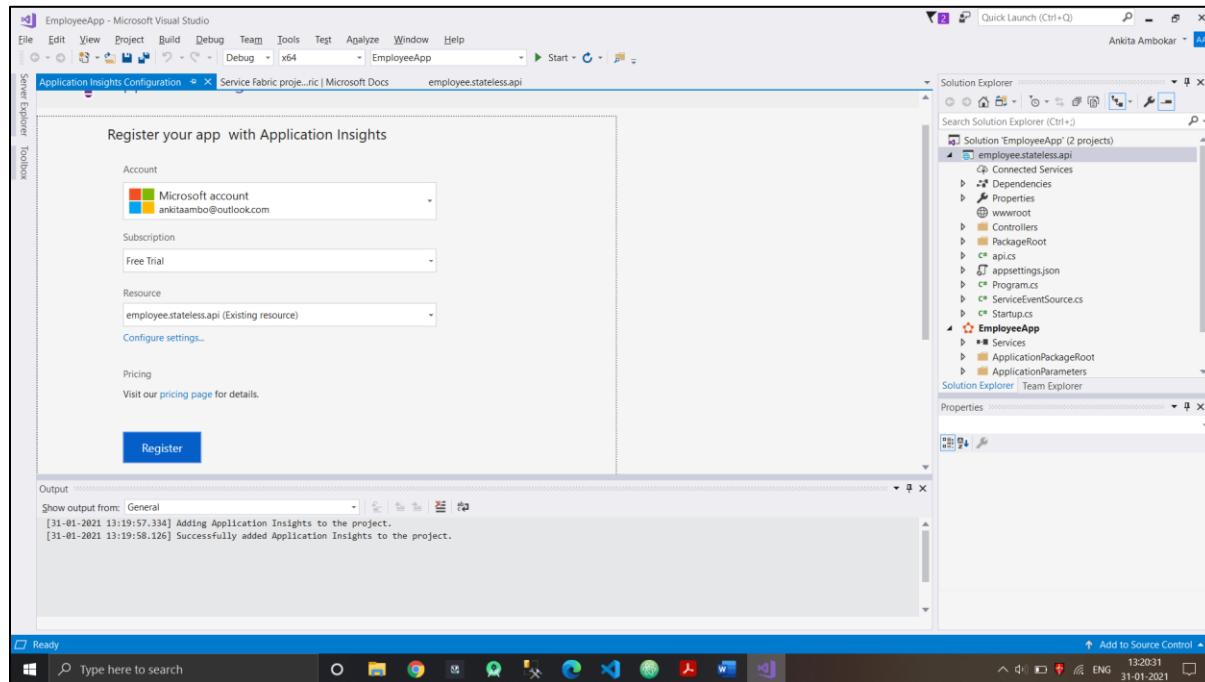
8. Choose Monitoring with Application Insights, as seen in Figure.



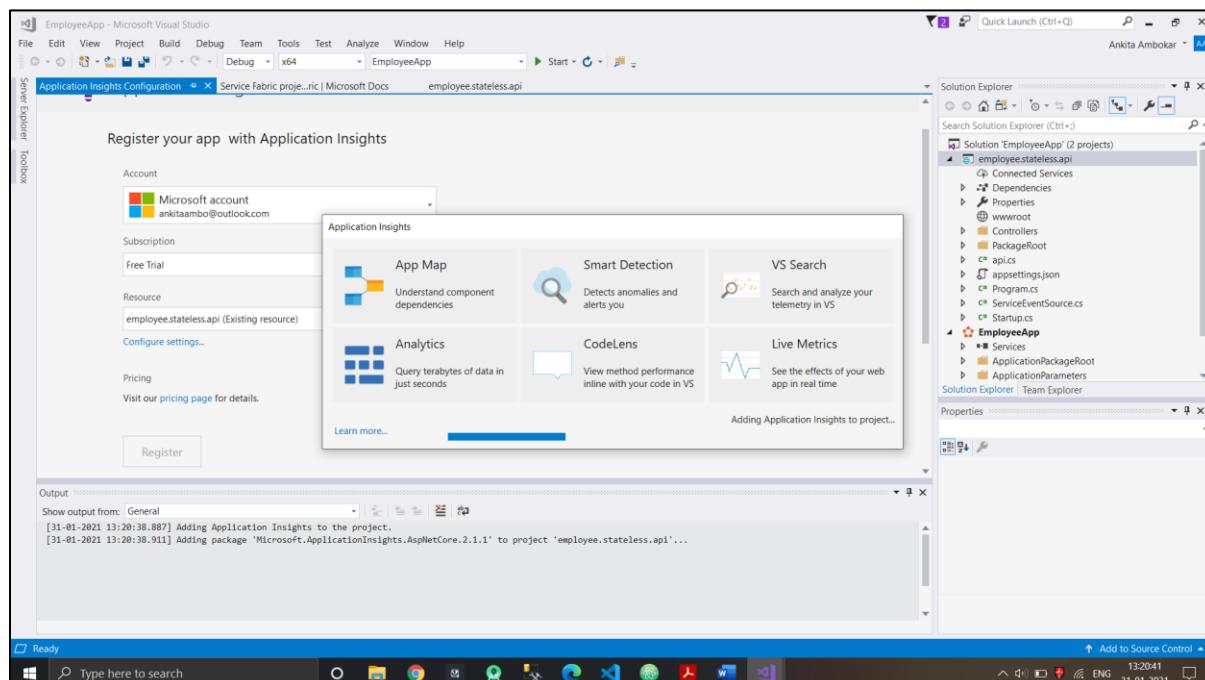
9. Click Get Started, as seen in Figure.



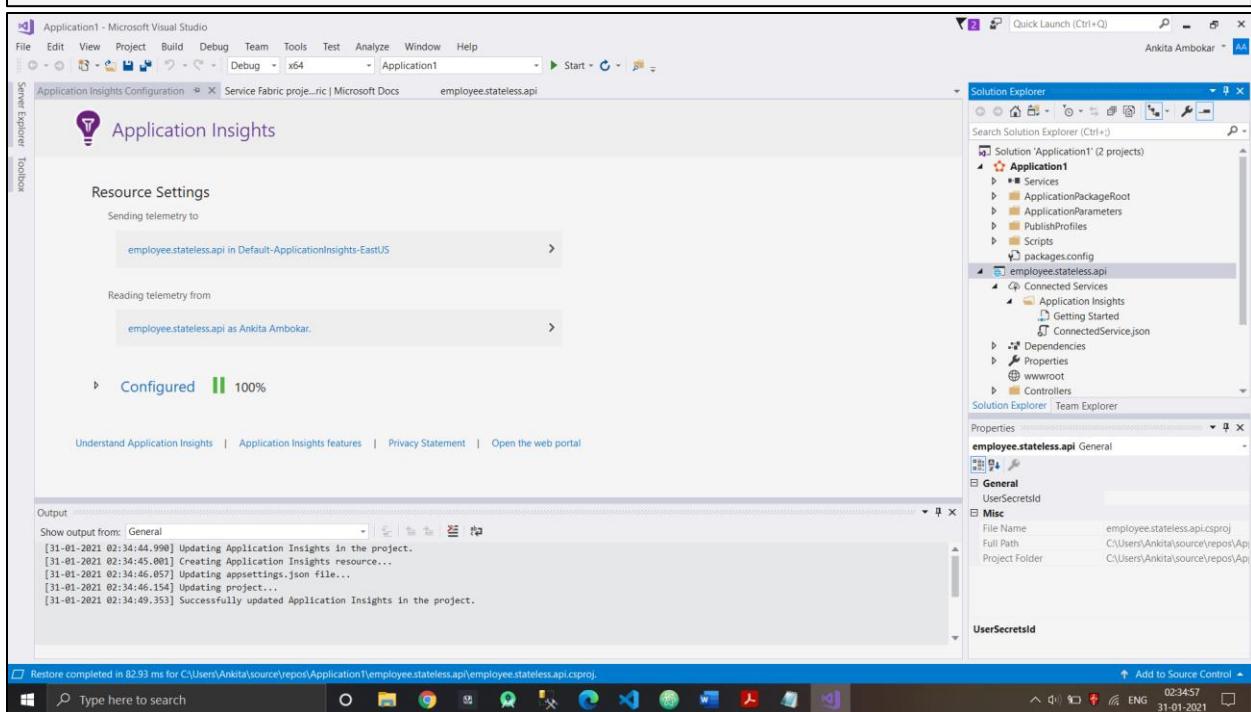
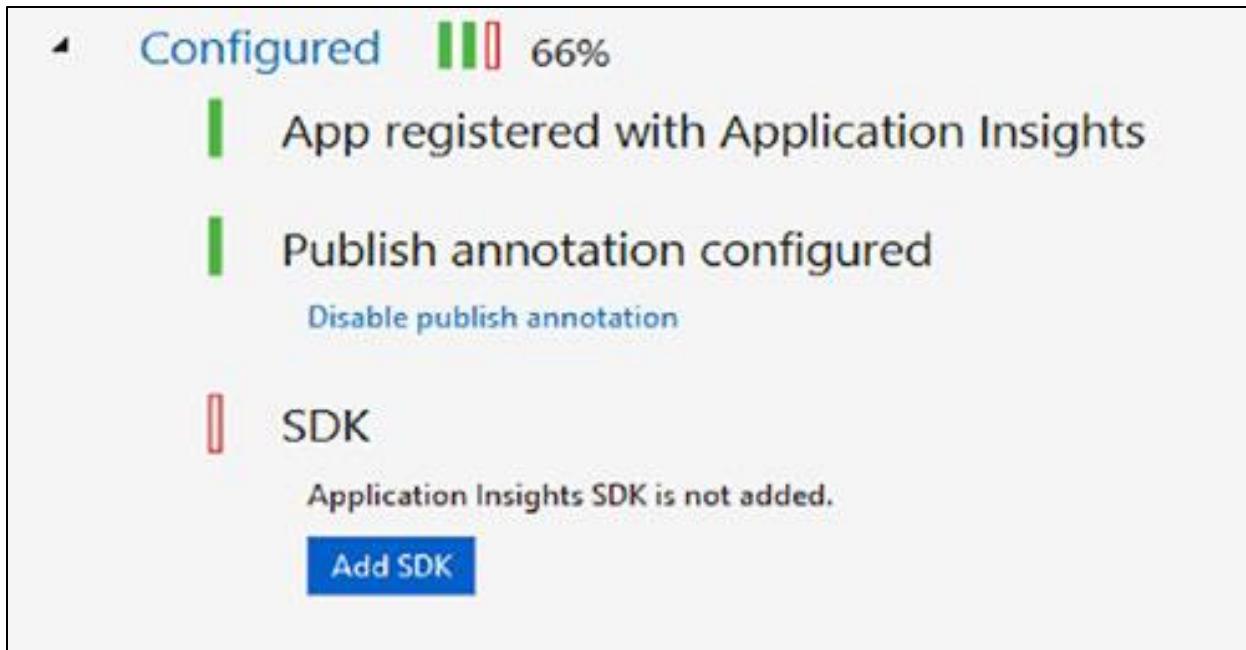
Choose the right Azure subscription and Application Insights resource. Once done, click Register, as seen in Figure.



It takes a few minutes to create the Application Insights resource in your Azure subscription. During the registration process, you see the screen shown in Figure.



11. Once the Application Insights configuration is complete, you see the status as 100%. If you see the Add SDK button (as shown in Figure), click it to achieve 100% status, as seen in Figure.



12. To confirm the Application Insights configuration, check the instrumentation key in appsettings.json.

13. Right-click the employee.stateless.api project to add dependencies for the following NuGet packages.

- Microsoft.EntityFrameworkCore.SqlServer
- Microsoft.ApplicationInsights.ServiceFabric.Native
- Microsoft.ApplicationInsights.AspNetCore

We are done with the configuration. Now let's add EmployeeController.

1. Right-click the employee.stateless.api project and a folder called Models. Add the following classes from the sources folder.

- AppSettings.cs
- Employee.cs
- SampleContext.cs
- TranslationResponse.cs

2. Right-click the employee.stateless.api project and add a file named DbInitializer.cs. Replace that content with the following content.

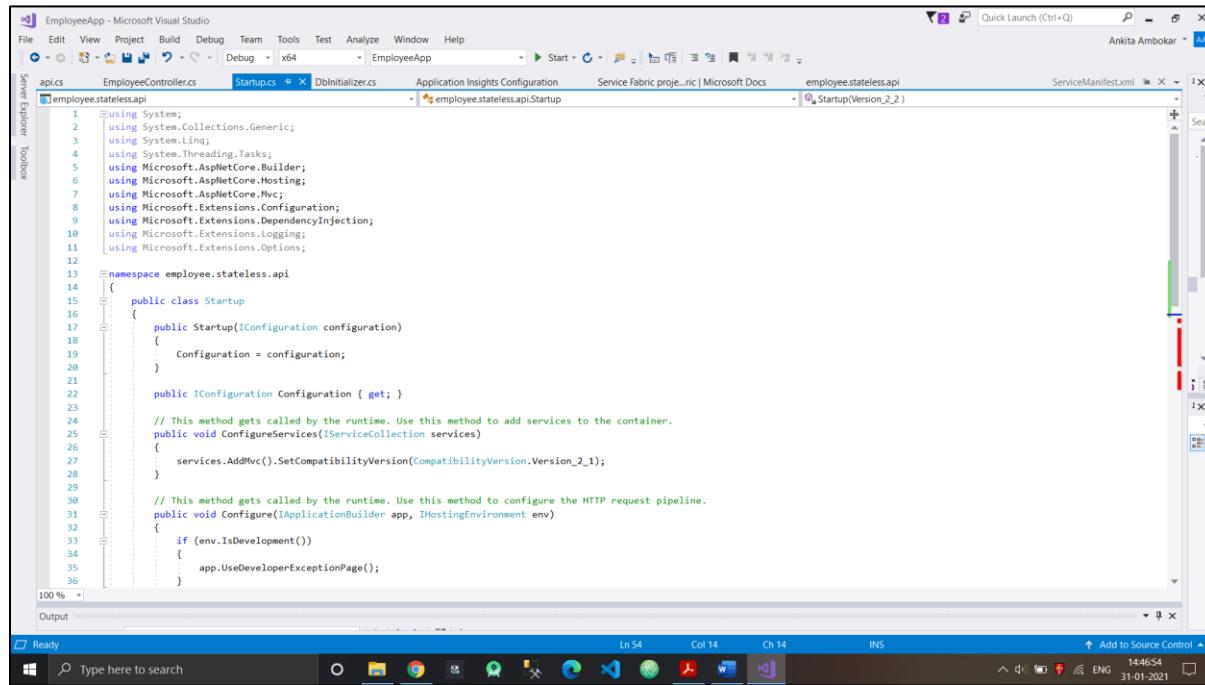
```
EmployeeApp - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
EmployeeController.cs Startup.cs DbInitializer.cs api.cs Application Insights Configuration Service Fabric project | Microsoft Docs employee.stateless.api ServiceManifest.xml
EmployeeController.cs
DbInitializer.cs
api.cs
employee.stateless.api
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Threading.Tasks;
5
6 namespace employee.stateless.api
7 {
8     public class DbInitializer
9     {
10         private SampleContext _context = null;
11         public DbInitializer(SampleContext context)
12         {
13             _context = context;
14         }
15         public void Initialize()
16         {
17             _context.Database.EnsureCreated();
18         }
19     }
20 }
```

3. Open Api.cs and replace the contents of the CreateServiceInstanceListeners method with the following content.

```
EmployeeApp - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
EmployeeController.cs Startup.cs DbInitializer.cs Application Insights Configuration Service Fabric project | Microsoft Docs employee.stateless.api ServiceManifest.xml
EmployeeController.cs
Startup.cs
DbInitializer.cs
api.cs
employee.stateless.api
1 using System.Collections.Generic;
2 using System.Fabric;
3 using System.IO;
4 using Microsoft.AspNetCore.Hosting;
5 using Microsoft.Extensions.DependencyInjection;
6 using Microsoft.ServiceFabric.Services.Communication.AspNetCore;
7 using Microsoft.ServiceFabric.Services.Communication.Runtime;
8 using Microsoft.ServiceFabric.Services.Runtime;
9 using Microsoft.Extensions.Configuration;
10 using Microsoft.ApplicationInsights.Extensibility;
11 using Microsoft.ApplicationInsights.ServiceFabric;
12
13 namespace employee.stateless.api
14 {
15     /// <summary>
16     /// The FabricRuntime creates an instance of this class for each service type instance.
17     /// </summary>
18     internal sealed class api : StatelessService
19     {
20         public api(StatelessServiceContext context)
21         : base(context)
22     }
23
24     /// <summary>
25     /// Optional override to create listeners (like tcp, http) for this service instance.
26     /// </summary>
27     /// <returns>The collection of listeners.</returns>
28     /// <returns>The collection of listeners.</returns>
29     protected override IEnumerable<ServiceInstanceListener> CreateServiceInstanceListeners()
30     {
31         return new ServiceInstanceListener[]
32         {
33             new KestrelCommunicationListener(serviceContext,
34             new KestrelCommunicationListener(serviceContext,
35             Context, "ServiceEndpoint", (url, listener) =>
36         {
100 %
```

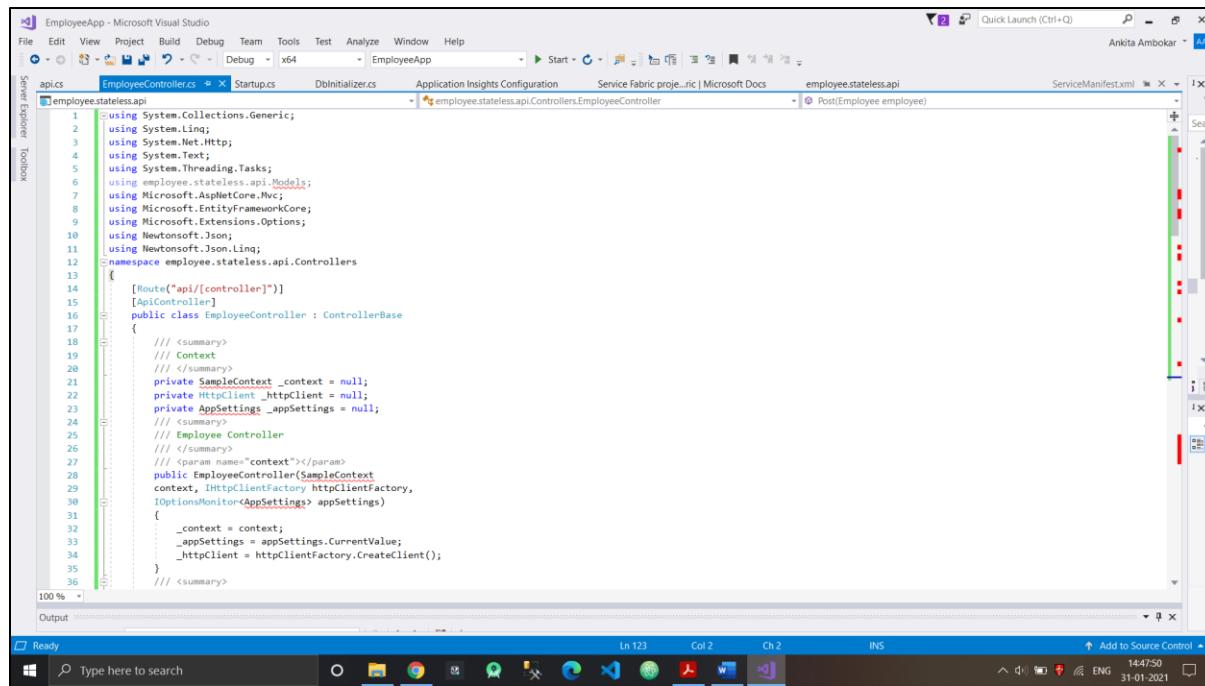
```
EmployeeApp - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
EmployeeController.cs Startup.cs DbInitializer.cs Application Insights Configuration Service Fabric project | Microsoft Docs employee.stateless.api ServiceManifest.xml
EmployeeController.cs
Startup.cs
DbInitializer.cs
api.cs
employee.stateless.api
49     optional: false,
50     reloadOnChange:
51     true
52   })
53   .ConfigureServices(
54     services => services
55     .AddSingleton
56     <StatelessService
57     Context>(service
58     Context,
59     integration
60     .AddSingleton
61     <ITelemetry
62     Initializer>
63     ((serviceProvider)
64     => FabricTelemetry
65     Initializer
66     .CreateFabric
67     Extension.
68     CreateFabric
69     Telemetry
70     Initializer
71     (serviceContext))
72     .UseContentRoot(Directory.
73     GetContentRoot())
74     .UseStartupConfig()
75     .UseApplicationInsights()
76     .UseServiceFabricIntegration
77     (listener, ServiceFabric
78     IntegrationOptions.None)
79     .UseUrls(url)
80     .Build());
81   );
82 }
83 }
```

4. Open Startup.cs and replace the contents of the ConfigureServices method with the following content.



```
EmployeeApp - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
... EmployeeApp ... Start ... x64 ... EmployeeApp ... Start ... Application Insights Configuration Service Fabric proj... Microsoft Docs employee.stateless.api ServiceManifest.xml ...
Server Explorer Toolbox
api.cs EmployeeController.cs Startup.cs DblInitializer.cs Application Insights Configuration Service Fabric proj... Microsoft Docs employee.stateless.api ServiceManifest.xml ...
employee.stateless.api
1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Threading.Tasks;
5 using Microsoft.AspNetCore.Builder;
6 using Microsoft.AspNetCore.Hosting;
7 using Microsoft.AspNetCore.Mvc;
8 using Microsoft.Extensions.Configuration;
9 using Microsoft.Extensions.DependencyInjection;
10 using Microsoft.Extensions.Logging;
11 using Microsoft.Extensions.Options;
12
13 namespace employee.stateless.api
14 {
15     public class Startup
16     {
17         public Startup(IConfiguration configuration)
18         {
19             Configuration = configuration;
20         }
21
22         public IConfiguration Configuration { get; }
23
24         // This method gets called by the runtime. Use this method to add services to the container.
25         public void ConfigureServices(IServiceCollection services)
26         {
27             services.AddMvc().SetCompatibilityVersion(CompatibilityVersion.Version_2_1);
28         }
29
30         // This method gets called by the runtime. Use this method to configure the HTTP request pipeline.
31         public void Configure(IApplicationBuilder app, IHostingEnvironment env)
32         {
33             if (env.IsDevelopment())
34             {
35                 app.UseDeveloperExceptionPage();
36             }
37         }
38     }
39 }
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100 % Output
```

5. Right-click the controller folder in the employee.stateless.api project and add a controller called EmployeeController.cs. Replace that content with the following content.



```
EmployeeApp - Microsoft Visual Studio
File Edit View Project Build Debug Team Tools Test Analyze Window Help
... EmployeeApp ... Start ... x64 ... EmployeeApp ... Start ... Application Insights Configuration Service Fabric proj... Microsoft Docs employee.stateless.api ServiceManifest.xml ...
Server Explorer Toolbox
api.cs EmployeeController.cs Startup.cs DblInitializer.cs Application Insights Configuration Service Fabric proj... Microsoft Docs employee.stateless.api ServiceManifest.xml ...
employee.stateless.api.Controllers.EmployeeController
1 using System.Collections.Generic;
2 using System.Linq;
3 using System.Net.Http;
4 using System.Text;
5 using System.Threading.Tasks;
6 using employee.stateless.api.Models;
7 using Microsoft.AspNetCore.Mvc;
8 using Microsoft.EntityFrameworkCore;
9 using Microsoft.Extensions.Options;
10 using Newtonsoft.Json;
11 using Newtonsoft.Json.Linq;
12
13 namespace employee.stateless.api.Controllers
14 {
15     [Route("api/[controller]")]
16     [ApiController]
17     public class EmployeeController : ControllerBase
18     {
19         /// <summary>
20         /// Context
21         /// </summary>
22         private SampleContext _context = null;
23         private HttpClient _httpClient = null;
24         private AppSettings _appSettings = null;
25         /// <summary>
26         /// Employee Controller
27         /// </summary>
28         /// <param name="context"></param>
29         public EmployeeController(SampleContext context, IHttpClientFactory httpClientFactory, IOptionsMonitor<AppSettings> appSettings)
30         {
31             _context = context;
32             _appSettings = appSettings.CurrentValue;
33             _httpClient = httpClientFactory.CreateClient();
34         }
35         /// <summary>
36     }
37 }
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100 % Output
```

```

apics EmployeeController.cs Startup.cs DblInitializer.cs Application Insights Configuration Service Fabric proje...xml | Microsoft Docs employee.stateless.api ServiceManifest.xml
  52  /// <returns></returns>
  53  [HttpGet("{id}")]
  54  public async Task<ActionResult> GetEmployee(int id)
  55  {
  56      Employee employee = await _context.Employees.
  57      Where(e => e.Id.Equals(id)).FirstOrDefaultAsync();
  58      return new OkObjectResult(employee);
  59  }
  60
  61  /// <summary>
  62  /// Creates an employee
  63  /// </summary>
  64  /// <param name="employee"></param>
  65  /// <returns></returns>
  66  [HttpPost]
  67  public async Task<ActionResult> Post([FromBody]Employee employee)
  68  {
  69      employee.NativeLanguageName = await
  70      GetTranslatedText(employee.FirstName);
  71      await _context.Employees.AddAsync(employee);
  72      await _context.SaveChangesAsync();
  73      return new OkResult();
  74  }
  75
  76  /// <summary>
  77  /// Deletes the employee based on id
  78  /// </summary>
  79  /// <param name="id"></param>
  80  /// <returns></returns>
  81  [HttpDelete("{id}")]
  82  public async Task<ActionResult> Delete(int id)
  83  {
  84      Employee employee = await _context.Employees.
  85      Where(e => e.Id.Equals(id)).FirstOrDefaultAsync();
  86      if (employee == null)
  87      {
  88          return new NotFoundResult();
  89      }
  90      _context.Employees.Remove(employee);
  91      await _context.SaveChangesAsync();
  92      return new OkResult();
  93  }
  94
  95  /// <summary>
  96  /// Gets the name in hindi
  97  /// </summary>
  98  /// <param name="name"></param>
  99  /// <returns></returns>
 100 private async Task<string> GetTranslatedText(string name)
 101 {
 102     System.Object[] body = new System.Object[] { new {
 103         Text = name
 104     } };
 105     var requestBody = JsonConvert.SerializeObject(body);
 106     StringContent content = new StringContent(
 107         requestBody, Encoding.UTF8, "application/json");
 108     _httpClient.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", _appSettings.AccessKey);
 109     var result = await _httpClient.PostAsync(
 110         $"{_appSettings.TranslationApiUrl}/translate/api-version=3.0 & q={name}&to=hin", content);
 111     result.EnsureSuccessStatusCode();
 112     string translatedJson = await result.Content.
 113     ReadAsStringAsync();
 114     TranslationResponse response = Newtonsoft.Json.
 115     JsonConvert.DeserializeObject<TranslationResponse>
 116     (JArray.Parse(translatedJson)[0].ToString());
 117     return response.translations[0].text;
 118 }
 119
 120 }
 121
 122 }
 123

```

```

apics EmployeeController.cs Startup.cs DblInitializer.cs Application Insights Configuration Service Fabric proje...xml | Microsoft Docs employee.stateless.api ServiceManifest.xml
  91     _context.Employees.Remove(employee);
  92     await _context.SaveChangesAsync();
  93     return new OkResult();
  94
  95
  96     /// <summary>
  97     /// Gets the name in hindi
  98     /// </summary>
  99     /// <param name="name"></param>
 100    /// <returns></returns>
 101    private async Task<string> GetTranslatedText(string name)
 102    {
 103        System.Object[] body = new System.Object[] { new {
 104            Text = name
 105        } };
 106        var requestBody = JsonConvert.SerializeObject(body);
 107        StringContent content = new StringContent(
 108            requestBody, Encoding.UTF8, "application/json");
 109        _httpClient.DefaultRequestHeaders.Add("Ocp-Apim-Subscription-Key", _appSettings.AccessKey);
 110        var result = await _httpClient.PostAsync(
 111            $"{_appSettings.TranslationApiUrl}/translate/api-version=3.0 & q={name}&to=hin", content);
 112        result.EnsureSuccessStatusCode();
 113        string translatedJson = await result.Content.
 114        ReadAsStringAsync();
 115        TranslationResponse response = Newtonsoft.Json.
 116        JsonConvert.DeserializeObject<TranslationResponse>
 117        (JArray.Parse(translatedJson)[0].ToString());
 118        return response.translations[0].text;
 119    }
 120
 121 }
 122
 123

```

6. Open AppSettings.json and make sure that the content looks similar to your connection strings.

```

{
  "ConnectionStrings": {
    "DefaultConnection": "Server=<<YOUR_SERVER>>;Database=<<YOUR_DATABASE>>;User
ID=<<USER_ID>>;Password=<<PASSWORD>>;
    Trusted_Connection=False;Encrypt=True;MultipleActiveResult
    Sets=True;"
  },
  "AppSettings": {
    "TranslationApiUrl": "https://api.cognitive.
    microsofttranslator.com",
  }
}

```

```
"AccessKey": "<<YOUR ACCESS KEY TO TRANSLATION API>>"  
},  
"Logging": {  
    "LogLevel": {  
        "Default": "Warning"  
    }  
},  
"AllowedHosts": "*",  
"ApplicationInsights": {  
    "InstrumentationKey": "<<YOUR INSTRUMENTATION KEY OF YOUR  
APP INSIGHTS RESOURCE>>"  
}
```

Practical No: 4

Practical No.4A Create an Azure Kubernetes Service Cluster

Steps –

1. Sign in Azure Portal (<https://portal.azure.com/>)
2. Create Resource by clicking on Create Resource option.
3. Select Kubernetes Services
4. Enter the subscription, resource group, kubernetes, Cluster name, Region, Kubernetes version,& DNS Name prefix
5. Click on Review + Create Button
6. Click on Create Button

The screenshot shows the 'Create Kubernetes cluster' wizard. At the top, there's a breadcrumb navigation: Home > New > Create Kubernetes cluster. Below that is a navigation bar with tabs: Basics (which is selected), Node pools, Authentication, Networking, Integrations, Tags, and Review + create. The Basics tab contains a descriptive paragraph about AKS and a 'Project details' section. In 'Project details', there are dropdown menus for 'Subscription' (Free Trial) and 'Resource group' ((New) CAD). Below these are sections for 'Cluster details': 'Kubernetes cluster name' (CAD), 'Region' ((US) East US), 'Availability zones' (Zones 1,2,3), and 'Kubernetes version' (1.18.14 (default)). Under 'Primary node pool', there's a note about the number and size of nodes. At the bottom of the page are buttons for 'Review + create' (highlighted in blue), '< Previous', and 'Next : Node pools >'.

Practical No.4B - Enable Azure Dev Space on an AKS Cluster

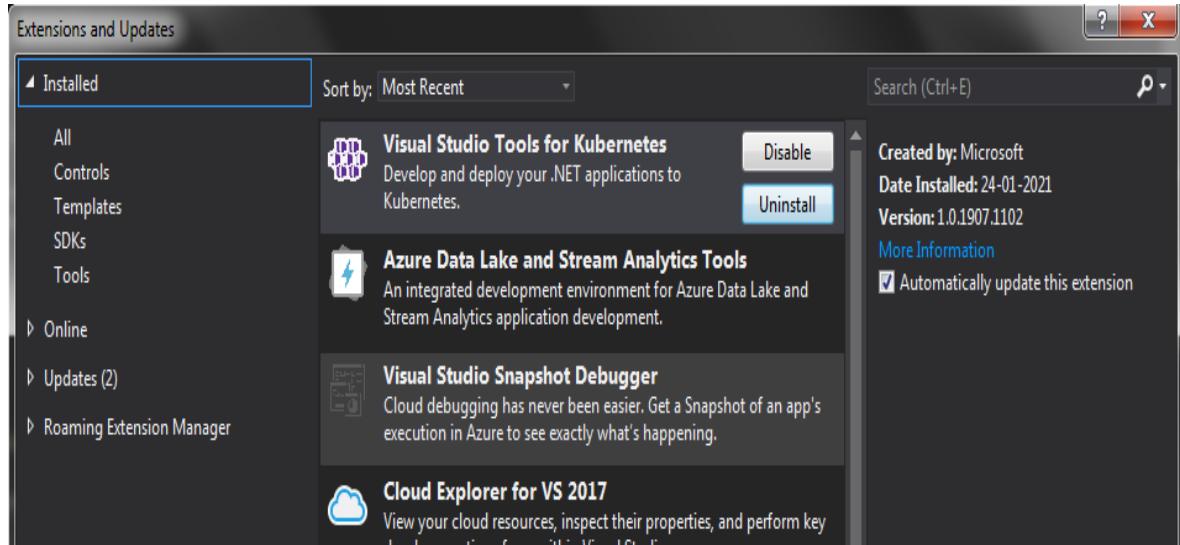
Steps –

1. Open Kubernetes Services
2. Click on Dev Space
3. Install Azure CLI
4. Open CMD
5. Enter command **AZ LOGIN**
6. Enter command
az aks use-dev-spaces -n CAD-g CAD
7. Install Azure Dev Space

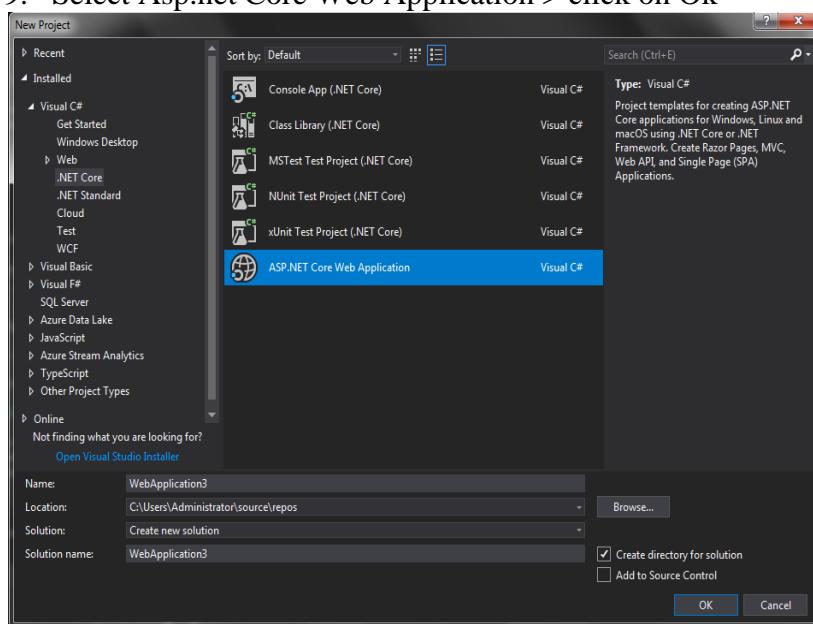
Practical No.4C - Configure Visual Studio to work with an azure kubernetes services Cluster

Steps –

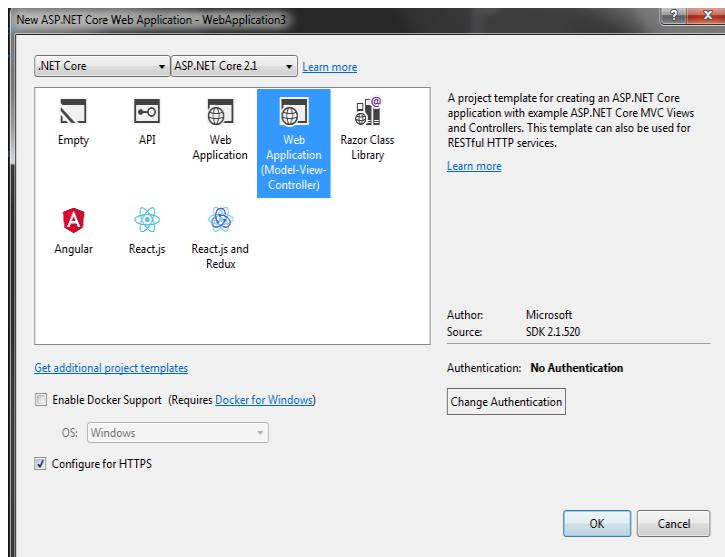
1. Install Visual Studio Enterprise 2017
2. Open Visual Studio 2017 > Tool > Extensions & Update
3. Search for Kubernetes Services tool
4. Click on Download
5. Close visual studio
6. click on Modify(install kubernetes tool extension)



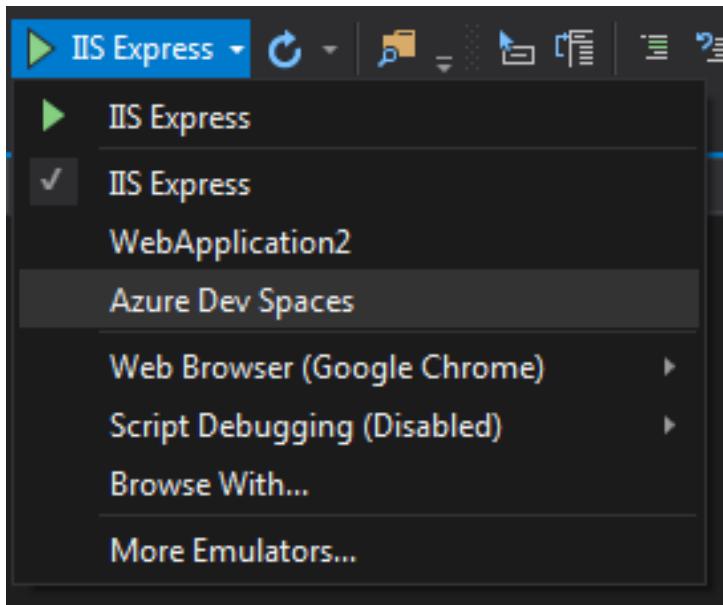
7. Open Visual studio
8. Click on new project
9. Select Asp.net Core Web Application > click on Ok



10. Select Model View Control > Click on OK



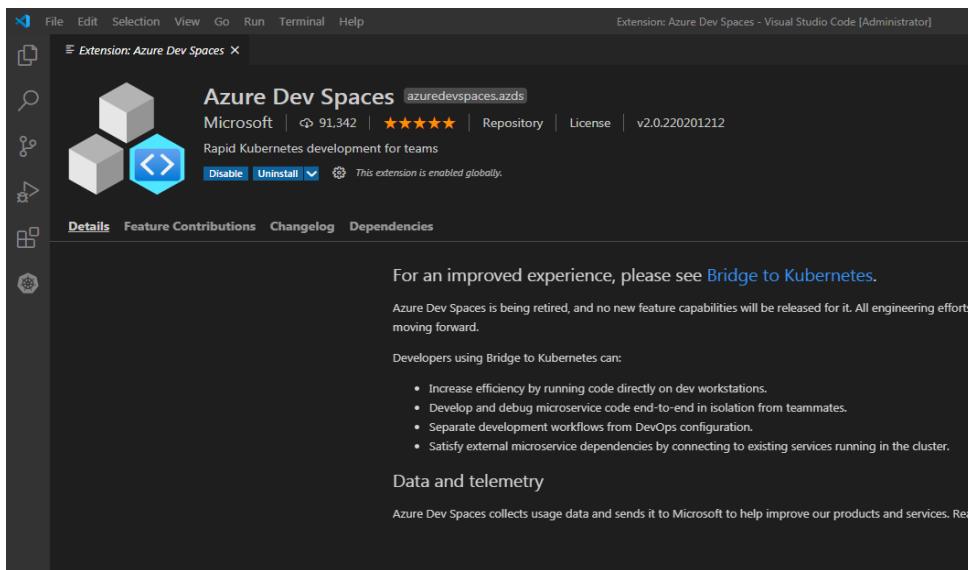
11. Click On Down Arrow of IIS Express
12. We can see option of Azure Dev Space



Practical No.4D - Configure Visual Studio Code to work with an azure kubernetes services Cluster

Steps –

1. Install visual studio code
2. Open extension window
3. Search for Azure Dev Space
4. Click Install



5. Install Azure CLI (<https://docs.microsoft.com/en-us/cli/azure/install-azure-cli?view=azure-cli-latest>)
6. Open cmd
7. Enter command **az Login**
8. Enter Command to install AZDS utility
az aks use-dev-spaces -n CAD -g CAD

```

Select Administrator: C:\Windows\system32\cmd.exe - az aks use-dev-spaces -n CLOUDAPP -g C...
Microsoft Windows [Version 6.1.7601]
Copyright (c) 2009 Microsoft Corporation. All rights reserved.

C:\Users\Administrator>az login
The default web browser has been opened at https://login.microsoftonline.com/con
non/oauth2/authorize. Please continue the login in the web browser. If no web br
oser is available or if the web browser fails to open, use device code flow wit
h 'az login --use-device-code'.
You have logged in. Now let us find all the subscriptions to which you have acce
ss...
[ {
  "cloudName": "AzureCloud",
  "homeTenantId": "6aabfd46-4740-4963-abed-282dabe77cde",
  "id": "1f62cf0-e8a8-4d07-8dc7-43c1d565c449",
  "isDefault": true,
  "managedByTenants": [],
  "name": "Free Trial",
  "state": "Enabled",
  "tenantId": "6aabfd46-4740-4963-abed-282dabe77cde",
  "user": {
    "name": "deepaliwalanju9920@gmail.com",
    "type": "user"
  }
}
]

C:\Users\Administrator>az aks use-dev-spaces -n CAD -g CAD
This command has been deprecated and will be removed in a future release.
For more information, please see https://github.com/Azure/dev-spaces/issues/418
Installing Dev Spaces commands...
A separate window will open to guide you through the installation process.

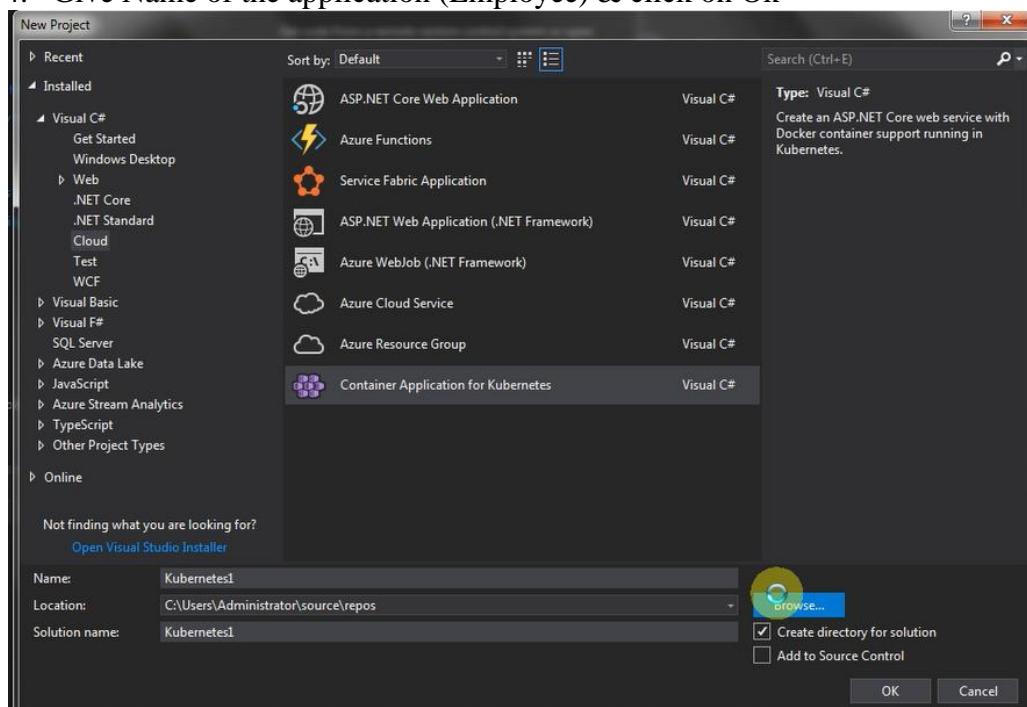
```

Practical No.4E - Deploy Application on AKS

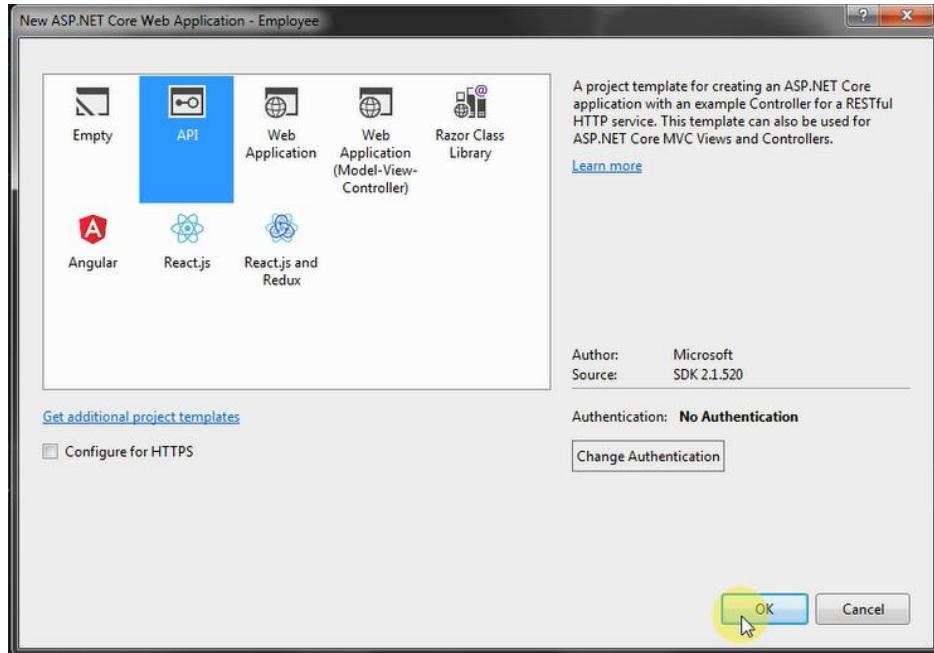
1.Core web API

Steps –

1. Open Visual Studio
2. Click on New project
3. Select Cloud Container Application for Kubernetes
4. Give Name of the application (Employee) & click on Ok



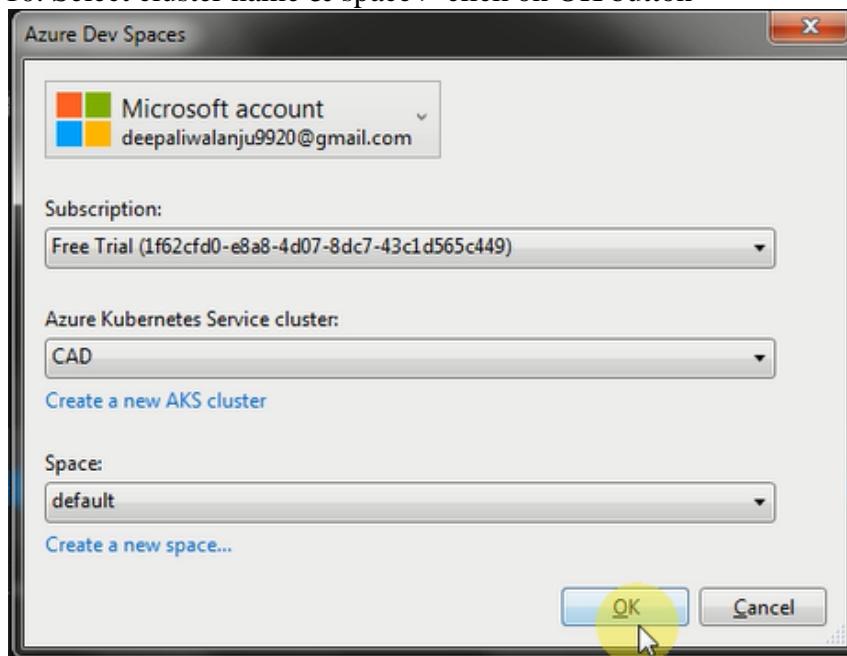
5. Select API option & click on OK button



6. Make sure Azure Dev Space is selected in menu.
7. Open ValuesController.cs file
8. Edit line number 16 as shown in the screen shot

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Threading.Tasks;
5 using Microsoft.AspNetCore.Mvc;
6
7 namespace EMPLOYEE.Controllers
8 {
9 [Route("api/[controller]")]
10 [ApiController]
11 public class ValuesController : ControllerBase
12 {
13 // GET api/values
14 [HttpGet]
15 public ActionResult<IEnumerable<string>> Get()
16 {
17 return new string[] { "Azure", "Kubernetes", "Service" };
18 }

9. Save changes & press F5 button to run the application
10. Select cluster name & space > click on OK button



Output

Show output from: Azure Dev Spaces

```

NAMESPACE: default
STATUS: DEPLOYED
RESOURCES:
==> v1/Service
NAME      TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
employee   ClusterIP   10.0.192.15   <none>        80/TCP      0s
==> v1beta2/Deployment
NAME      DESIRED  CURRENT  UP-TO-DATE  AVAILABLE  AGE
employee  1         0         0          0          0s
==> v1beta1/Ingress
NAME      HOSTS      ADDRESS      PORTS      AGE
employee  default.employee.jpksngkw2j.eus.azds.io  13.68.220.220  80      0s
==> v1/Pod(related)
NAME      READY  STATUS      RESTARTS  AGE
employee-6cf9f9b99-z1v4r  0/2    Pending    0          0s
NOTES:
1. Get the application URL by running these commands:
http://default.employee.jpksngkw2j.eus.azds.io/
Building container image...
Sending build context to Docker daemon 17.92kB
Step 1/13 : FROM microsoft/aspnetcore-build:2.0
--> 06a6525397c2

```

2.Node.js API

Steps –

1. Create MYAPP folder
2. Open visual studio code
3. Open folder MYAPP in visual studio code
4. Create new file server.js & pacakge.json
5. From windows explorer create PUBLIC folder in MYAPP folder
6. Now create new file using public folder by using visual studio code
7. Create file index.html & app.css
8. Open command palette from view menu (ctrl+shif+p)
9. Enter azure dev space & click on it
10. Click on configure
11. Click DEBUG icon on left & then click on LAUNCH SERVER(AZDS)
12. OUTPUT will display in Output window

Server.js

```

var express = require('express');
var app = express();
app.use(express.static(__dirname + '/public'));
app.get('/', function (req, res) {
res.sendFile(__dirname + '/public/index.html');
});
app.get('/api', function (req, res) {
res.send('Hello from webfrontend');
});
var port = process.env.PORT || 80;
var server = app.listen(port, function () {
console.log('Listening on port ' + port);
});
process.on("SIGINT", () => {
process.exit(130 /* 128 + SIGINT */);
});
process.on("SIGTERM", () => {
console.log("Terminating...");
server.close();
});

```

Package.json

```
{
"name": "webfrontend",
"version": "0.1.0",
"devDependencies": {
"nodemon": "^1.18.10"
},
"dependencies": {
"express": "^4.16.2",

```

```

"request": "2.83.0"
},
"main": "server.js",
"scripts": {
"start": "node server.js"
}
}

```

Index.html

```

<!doctype html>
<html ng-app="myApp">
<head>
<script src="https://ajax.googleapis.com/ajax/libs/
angularjs/1.5.3/angular.min.js"></script>
<script src="https://cdnjs.cloudflare.com/ajax/
libs/angular.js/1.5.3/angular-route.js"></script>
<script src="app.js"></script>
<link rel="stylesheet" href="app.css">
<link href="https://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min.css"
rel="stylesheet" integrity="sha384-1q8mTJOASx8j1Au
+a5WDVnPi2lkFfwwEAa8hDDdjZlpLegxhjVME1fgjWPGmkzs7"
crossorigin="anonymous">
<!-- Uncomment the next line -->
<!-- <meta name="viewport" content="width=device-width,
initial-scale=1" -->
</head>
<body style="margin-left:10px; margin-right:10px;">
<div ng-controller="MainController">
<h2>Server Says</h2>
<div class="row">
<div class="col-xs-8 col-md-10">
<div ng-repeat="message in messages
track by $index">
<span class="message">{ message }</
span>
</div>
</div>
<div class="col-xs-4 col-md-2">
<button class="btn btn-primary"
ng-click="
sayHelloToServer()">Say It
Again</button>
</div>
</div>
</div>
</body>
</html>

```

APP.css

```

.message {
font-family: Courier New, Courier, monospace;
font-weight: bold;
}

```

OUTPUT

The Debug console shows the log output.

```

> Executing task: C:\Program Files\Microsoft SDKs\Azure\Azure
Dev Spaces CLI (Preview)\azds.exe up --port=50521:9229 --await-exec
--keep-alive <
Synchronizing files...4s

```

Using dev space 'new01' with target 'kuber01'
Installing Helm chart...2s
Waiting for container image build...29s
Building container image...
Step 1/8 : FROM node:lts
Step 2/8 : ENV PORT 80
Step 3/8 : EXPOSE 80
Step 4/8 : WORKDIR /app
Step 5/8 : COPY package.json .
Step 6/8 : RUN npm install
Step 7/8 : COPY ..
Step 8/8 : CMD ["npm", "start"]
Built container image in 45s
Waiting for container...52s
Service 'myapp' port 80 (http) is available via port forwarding
at http://localhost:50764
Terminal will be reused by tasks, press any key to close it.

PRACTICAL 5

Create an AKS cluster

a. From the portal

Step 1: Sign in to the Azure portal at <https://portal.azure.com>.

Step 2: On the Azure portal menu or from the Home page, select Create a resource.

The screenshot shows the Microsoft Azure portal's Home page. At the top, there's a navigation bar with 'Microsoft Azure', 'Upgrade', and a search bar. Below the navigation bar is a row of service icons: 'Create a resource', 'Virtual machines', 'App Services', 'Storage accounts', 'SQL databases', 'Azure Database for PostgreSQL...', 'Azure Cosmos DB', 'Kubernetes services', 'Function App', and 'More services'. Underneath these are sections for 'Navigate' (Subscriptions, Resource groups, All resources, Dashboard) and 'Tools' (Microsoft Learn, Azure Monitor, Security Center, Cost Management). There's also a 'Useful links' section with links to documentation, Azure Services, and recent updates. At the bottom, there's a search bar and a taskbar with various application icons.

Step 3: Select Containers > Kubernetes Service.

The screenshot shows the Azure portal Marketplace page. The 'Containers' category is highlighted with a blue border. Other categories like 'Compute' and 'Databases' are also visible. To the right, there are cards for 'Container Instances', 'Container Registry', 'Kubernetes Service', 'Web App for Containers', 'DC/OS on Azure (preview)', and 'WordPress Container Image (preview)'. At the bottom, there's a search bar and a taskbar with various application icons.

Step 4: On the Basics page, configure the following options:

- **Project details:** Select an Azure Subscription, then select or create an Azure Resource group, such as *myResourceGroup*.
- **Cluster details:** Enter a **Kubernetes cluster name**, such as *myCluster*. Select a **Region** and **Kubernetes version** for the AKS cluster.
- **Primary node pool:** Select a VM **Node size** for the AKS nodes. The VM size *can't* be changed once an AKS cluster has been deployed. - Select the number of nodes to deploy into the cluster. Set **Node count** to *1*. Node count *can* be adjusted after the cluster has been deployed.

Select Next: Node pools when complete.

Project details
Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * Free Trial
Resource group * (New) myResourceGroup
[Create new](#)

Cluster details
Kubernetes cluster name * myCluster
Region * (Asia Pacific) Central India
Availability zones None
No availability zones are available for the location you have selected.
[View locations that support availability zones](#)
Kubernetes version * 1.18.14 (default)

Review + create < Previous Next : Node pools >

Step 5: On the **Node pools** page, keep the default options. At the bottom of the screen, click **Next: Authentication**.

Node pools
In addition to the required primary node pool configured on the Basics tab, you can also add optional node pools to handle a variety of workloads [Learn more about multiple node pools](#)

| Name | Mode | OS type | Node count | Node size |
|-----------|--------|---------|------------|-----------------|
| agentpool | System | Linux | 3 | Standard_DS2_v2 |

Enable virtual nodes
Virtual nodes allow burstable scaling backed by serverless Azure Container Instances. [Learn more about virtual nodes](#)

Enable virtual nodes Virtual nodes are not available in 'Central India'. Supported regions are: eastus2euap, westcentralus, centraluseuap, westus, westeurope, australiaeast, eastus, japaneast, northeurope, southeastasia, eastus2, westus2, centralus, southcentralus, canadacentral, koreacentral, francecentral

Review + create < Previous Next : Authentication >

Step 6: On the **Authentication** page, configure the following options:

- Create a new service principal by leaving the **Service Principal** field with **(new) default service principal**. Or you can choose *Configure service principal* to use an existing one. If you use an existing one, you will need to provide the SPN client ID and secret.
- Enable the option for Kubernetes role-based access control (Kubernetes RBAC). This will provide more fine-grained control over access to the Kubernetes resources deployed in your AKS cluster

Cluster infrastructure
The cluster infrastructure authentication specified is used by Azure Kubernetes Service to manage cloud resources attached to the cluster. This can be either a [service principal](#) or a [system-assigned managed identity](#).

Authentication method Service principal System-assigned managed identity
Service principal * 139935fa-c850-4a40-8424-cfb75c392006
[Configure service principal](#)

Kubernetes authentication and authorization
Authentication and authorization are used by the Kubernetes cluster to control user access to the cluster as well as what the user may do once authenticated. [Learn more about Kubernetes authentication](#)

Role-based access control (RBAC) Enabled Disabled
AKS-managed Azure Active Directory Enabled Disabled

Node pool OS disk encryption
By default, all disks in AKS are encrypted at rest with Microsoft-managed keys. For additional control over encryption, you can [supply your own keys using a disk encryption set backed by an Azure Key Vault](#). The disk encryption set will be used to...

Review + create < Previous Next : Networking >

Step 7: By default, *Basic networking, Integrations and tags* is used, and Azure

Monitor for containers is enabled.

Home > New >

Create Kubernetes cluster

You can change networking settings for your cluster, including enabling HTTP application routing and configuring your network using either the 'Kubenet' or 'Azure CNI' options:

- The **Kubenet** networking plug-in creates a new VNet for your cluster using default values.
- The **Azure CNI** networking plug-in allows clusters to use a new or existing VNet with customizable addresses. Application pods are connected directly to the VNet, which allows for native integration with VNet features.

Learn more about networking in Azure Kubernetes Service

Network configuration Kubenet Azure CNI

DNS name prefix * 

Traffic routing

Load balancer Standard

Enable HTTP application routing

[Review + create](#) [< Previous](#) [Next : Integrations >](#)

Home > New >

Create Kubernetes cluster

Connect your AKS cluster with additional services.

Azure Container Registry
Connect your cluster to an Azure Container Registry to enable seamless deployments from a private image registry. You can create a new registry or choose one you already have. [Learn more about Azure Container Registry](#)

Container registry 

 The system-assigned managed identity authentication method must be used in order to associate an Azure Container Registry.

Azure Monitor
In addition to the CPU and memory metrics included in AKS by default, you can enable Container Insights for more comprehensive data on the overall performance and health of your cluster. Billing is based on data ingestion and retention settings.
[Learn more about container performance and health monitoring](#)
[Learn more about pricing](#)

Container monitoring Enabled Disabled

[Review + create](#) [< Previous](#) [Next : Tags >](#)

Click Review + create and then Create when validation completes.

Microsoft Azure [Upgrade](#) [Search resources, services, and docs \(G+/-\)](#)

[Home](#) > [New](#) >

Create Kubernetes cluster

Validation passed

[Basics](#) [Node pools](#) [Authentication](#) [Networking](#) [Integrations](#) [Tags](#) [Review + create](#)

Basics

| | |
|-------------------------|-----------------|
| Subscription | Free Trial |
| Resource group | myResourceGroup |
| Region | Central India |
| Kubernetes cluster name | myCluster |
| Kubernetes version | 1.18.14 |

Node pools

| | |
|-----------------------------------|----------|
| Node pools | 1 |
| Enable virtual nodes | Disabled |
| Enable virtual machine scale sets | Enabled |

[Create](#) [< Previous](#) [Next >](#) [Download a template for automation](#)



Step 8: It takes a few minutes to create the AKS cluster. When your deployment is complete, click Go to resource, or browse to the AKS cluster resource group, such as *myResourceGroup*, and select the AKS resource, such as *myCluster*. The AKS cluster dashboard is shown below:

The screenshot shows the Microsoft Azure portal interface. At the top, there's a blue header bar with the Microsoft Azure logo, an 'Upgrade' button, and a search bar. Below the header, the URL 'microsoft.aks-20210121225543 | Overview' is displayed. On the left, there's a sidebar with navigation links: 'Overview' (which is selected and highlighted in grey), 'Inputs', 'Outputs', and 'Template'. The main content area has a heading 'Deployment is in progress'. Below it, a table titled 'Deployment details' shows two entries: 'myCluster' (Type: Microsoft.ContainerService..., Status: Created) and 'SolutionDeployment-202101' (Type: Microsoft.Resources/deploy..., Status: OK). A message at the bottom of the main content area says 'We'd love your feedback!'. To the right, there's a vertical sidebar with various Azure-related links like 'Container Insights', 'Free Microsoft Tutorials', and 'Work with an expert'. At the bottom, there's a taskbar with icons for different Windows applications.

This screenshot shows the same AKS cluster overview page after the deployment has completed. The status message now says 'Your deployment is complete'. The deployment details table remains the same. Below the table, under 'Next steps', there are three recommended actions: 'Create a Kubernetes deployment' (Recommended), 'Integrate automatic deployments within your cluster' (Recommended), and 'Connect to cluster' (Recommended). The rest of the interface is identical to the previous screenshot, including the sidebar and taskbar.

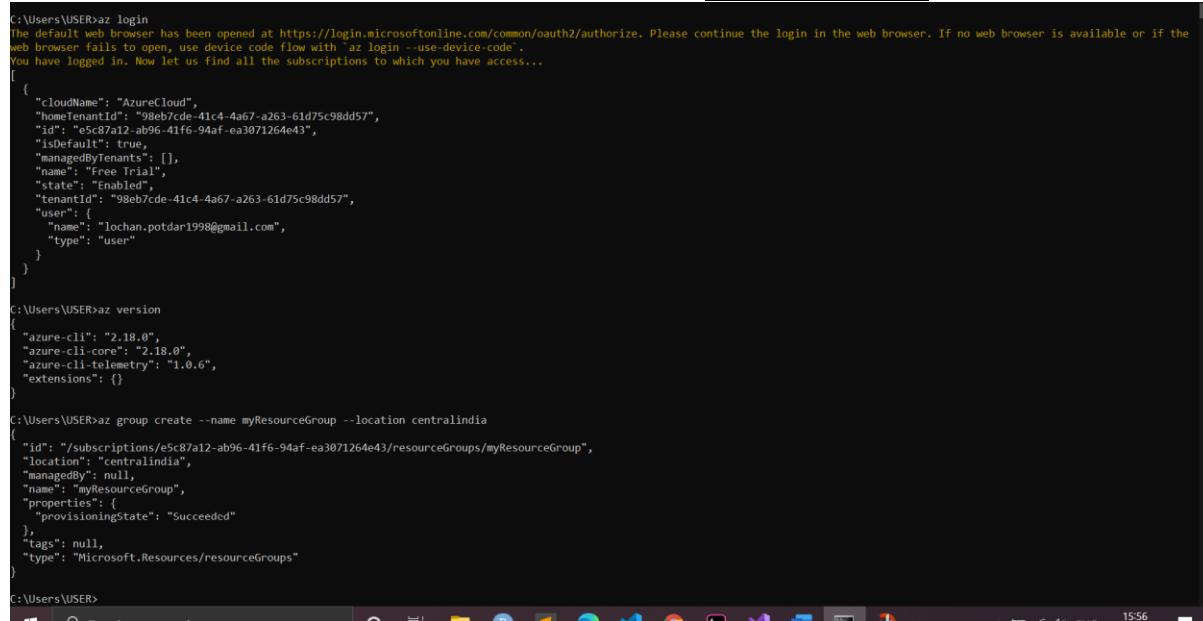
b. From azure CLI

Step 1: Install the Azure CLI to run CLI reference commands.

Step 2: Sign in to the Azure CLI by using the `az login` command. To finish the authentication process, follow the steps displayed in your terminal.

Step 3: Run `az version` to find the version and dependent libraries that are

installed. To upgrade to the latest version, run [az upgrade](#).



```
C:\Users\USER>az login
The default web browser has been opened at https://login.microsoftonline.com/common/oauth2/authorize. Please continue the login in the web browser. If no web browser is available or if the
web browser fails to open, use device code flow with 'az login --use-device-code'.
You have logged in. Now let us find all the subscriptions to which you have access...
[

  {
    "cloudName": "AzureCloud",
    "homeTenantId": "98eb7cde-41c4-4a67-a263-61d75c98dd57",
    "id": "e5c87a12-ab96-41f6-94af-ea3071264e43",
    "isDefault": true,
    "managedByTenants": [],
    "name": "Free Trial",
    "state": "Enabled",
    "tenantId": "98eb7cde-41c4-4a67-a263-61d75c98dd57",
    "user": {
      "name": "lochan.potdar1998@gmail.com",
      "type": "user"
    }
  }

C:\Users\USER>az version
{
  "azure-cli": "2.18.0",
  "azure-cli-core": "2.18.0",
  "azure-cli-telemetry": "1.0.6",
  "extensions": {}
}

C:\Users\USER>az group create --name myResourceGroup --location centralindia
{
  "id": "/subscriptions/e5c87a12-ab96-41f6-94af-ea3071264e43/resourceGroups/myResourceGroup",
  "location": "centralindia",
  "managedBy": null,
  "name": "myResourceGroup",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null,
  "type": "Microsoft.Resources/resourceGroups"
}

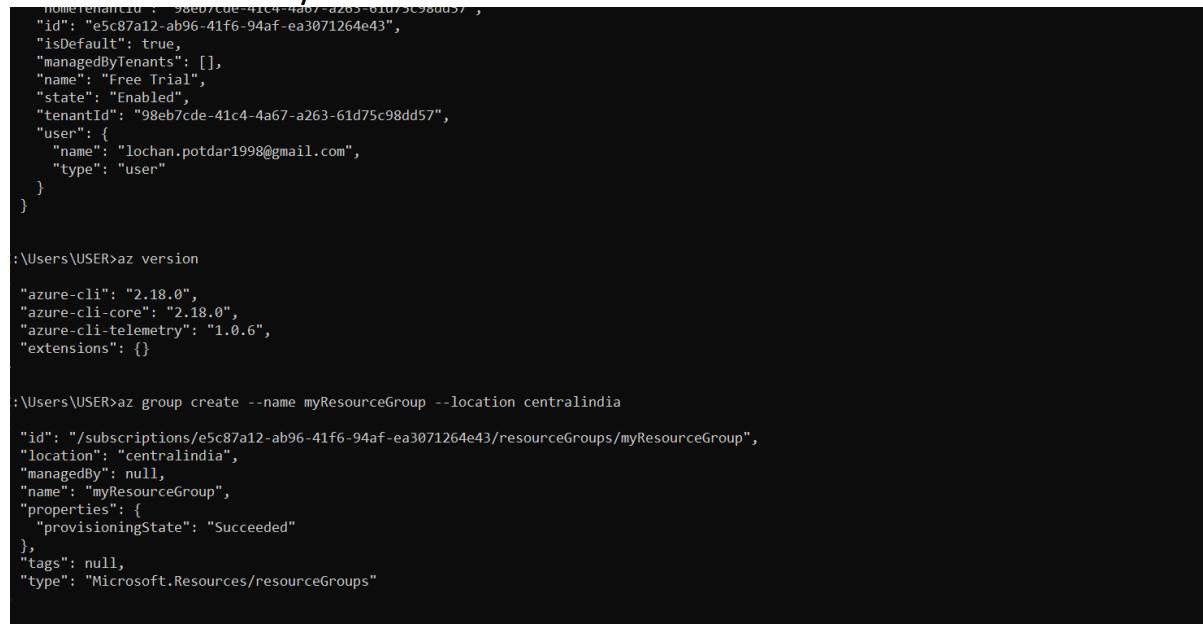
C:\Users\USER>
```

Step 4: An Azure resource group is a logical group in which Azure resources are deployed and managed. When you create a resource group, you are asked to specify a location. This location is where resource group metadata is stored, it is also where your resources run in Azure if you don't specify another region during resource creation. Create a resource group using the [az group create](#) command.

Step 5: The following example creates a resource group named myResourceGroup in the centralindia location.

```
az group create --name myResourceGroup --location centralindia
```

Output similar to the following example indicates the resource group has been created successfully:



```
:\\Users\\USER>az login
The default web browser has been opened at https://login.microsoftonline.com/common/oauth2/authorize. Please continue the login in the web browser. If no web browser is available or if the
web browser fails to open, use device code flow with 'az login --use-device-code'.
You have logged in. Now let us find all the subscriptions to which you have access...
[

  {
    "cloudName": "AzureCloud",
    "homeTenantId": "98eb7cde-41c4-4a67-a263-61d75c98dd57",
    "id": "e5c87a12-ab96-41f6-94af-ea3071264e43",
    "isDefault": true,
    "managedByTenants": [],
    "name": "Free Trial",
    "state": "Enabled",
    "tenantId": "98eb7cde-41c4-4a67-a263-61d75c98dd57",
    "user": {
      "name": "lochan.potdar1998@gmail.com",
      "type": "user"
    }
  }

:\\Users\\USER>az version
{
  "azure-cli": "2.18.0",
  "azure-cli-core": "2.18.0",
  "azure-cli-telemetry": "1.0.6",
  "extensions": {}

:\\Users\\USER>az group create --name myResourceGroup --location centralindia
{
  "id": "/subscriptions/e5c87a12-ab96-41f6-94af-ea3071264e43/resourceGroups/myResourceGroup",
  "location": "centralindia",
  "managedBy": null,
  "name": "myResourceGroup",
  "properties": {
    "provisioningState": "Succeeded"
  },
  "tags": null,
  "type": "Microsoft.Resources/resourceGroups"
}
```

Step 6: Use the [az aks create](#) command to create an AKS cluster. The

following example creates a cluster named *myAKSCluster* with one node. This will take several minutes to complete.

```
az aks create --resource-group myResourceGroup --name myAKSCluster --node-count 1 --enable-addons monitoring --generate-ssh-keys
```

```
C:\Users\USER>az aks create --resource-group myResourceGroup --name myAKSCluster --node-count 1 --enable-addons monitoring --generate-ssh-keys
AAD role propagation done[########################################] 100.0000%
{
  "aadProfile": null,
  "addonProfiles": {
    "kubeDashboard": {
      "config": null,
      "enabled": false,
      "identity": null
    },
    "omsagent": {
      "logAnalyticsWorkspaceResourceID": "/subscriptions/e5c87a12-ab96-41f6-94af-ea3071264e43/resourcegroups/defaultresourcegroup-cin/providers/microsoft.operationalinsights/workspaces/defaultworkspace-e5c87a12-ab96-41f6-94af-ea3071264e43-cin"
    },
    "enabled": true,
    "identity": {
      "clientId": "b9a41849-08a7-441a-ba76-eb84fe662c67",
      "objectID": "f1dec4f2-3662-4707-9d97-b5d8edbf48ff",
      "resourceId": "/subscriptions/e5c87a12-ab96-41f6-94af-ea3071264e43/resourcegroups/MC_myResourceGroup_myAKSCluster_centralindia/providers/Microsoft.ManagedIdentity/userAssignedIdentities/omsagent-myakscluster"
    }
  },
  "agentPoolProfiles": [
    {
      "availabilityZones": null,
      "count": 1,
      "enableAutoScaling": null,
      "enableNodePublicIP": false,
      "maxCount": null,
      "maxPods": 110,
      "minCount": null,
      "mode": "System",
      "name": "nodepool1",
      "nodeImageVersion": "AKSUbuntu-1804-2021.01.06",
      "nodeLabels": {},
      "nodeTaints": null,
      "orchestratorVersion": "1.18.14",
      "osDiskSizeGb": 128,
      "osDiskType": "Managed",
      "osType": "Linux",
      "powerState": {
        "code": "Running"
      },
      "vnetSubnetId": null
    }
  ],
  "nodeCount": 1
}
```

After a few minutes, the command completes and returns JSON-formatted information about the cluster.

Practical no 6

Create an Application Gateway Using Ocelot and Securing APIs with Azure AD

Q6: Create an Application Gateway Using Ocelot and Securing APIs with Azure AD.

API Gateway is an API management tool that usually sits between the external caller (Web or Mobile) and the internal services.

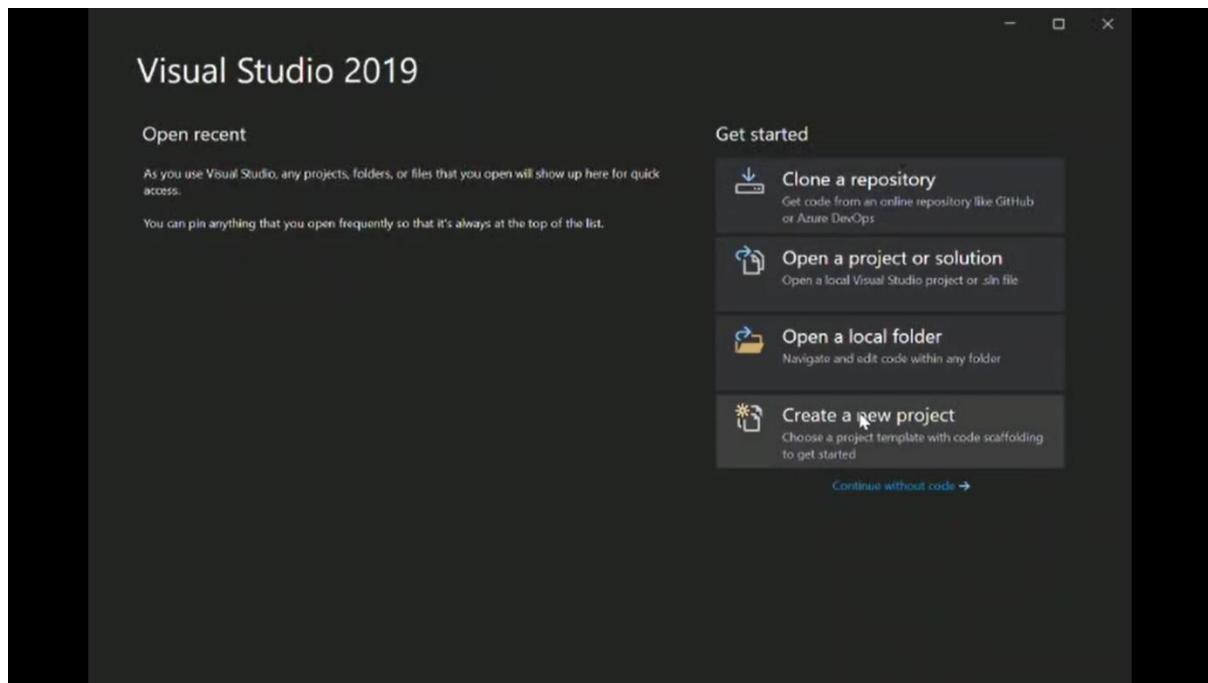
The API Gateway can provide multiple features like:

1. Routing Request
2. Aggregations
3. Authentication
4. Authorization
5. Rate Limiting
6. Caching
7. Load Balancing ETC.

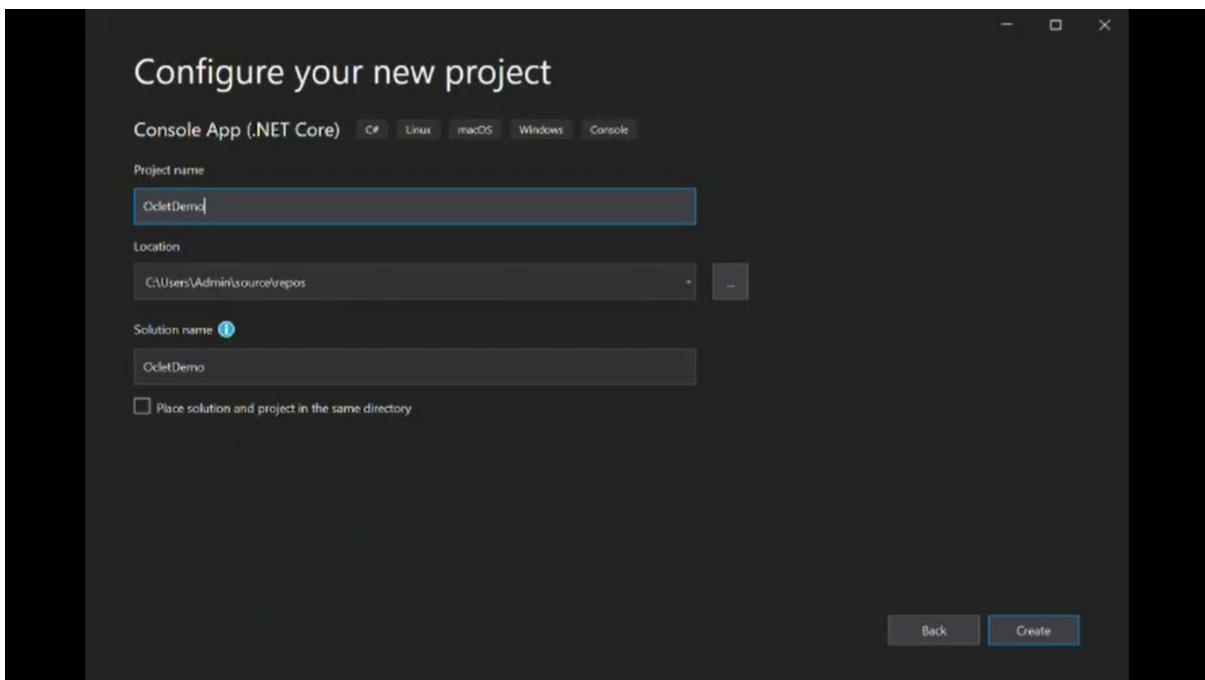
Ocelot is an ASP.Net Core (Supports .Net Core 3.1) API Gateway. It's a NuGet package, which can be added to any ASP.Net Core application to make it an API Gateway. Ocelot API Gateway supports all the features that any standard API Gateway does.

Steps:

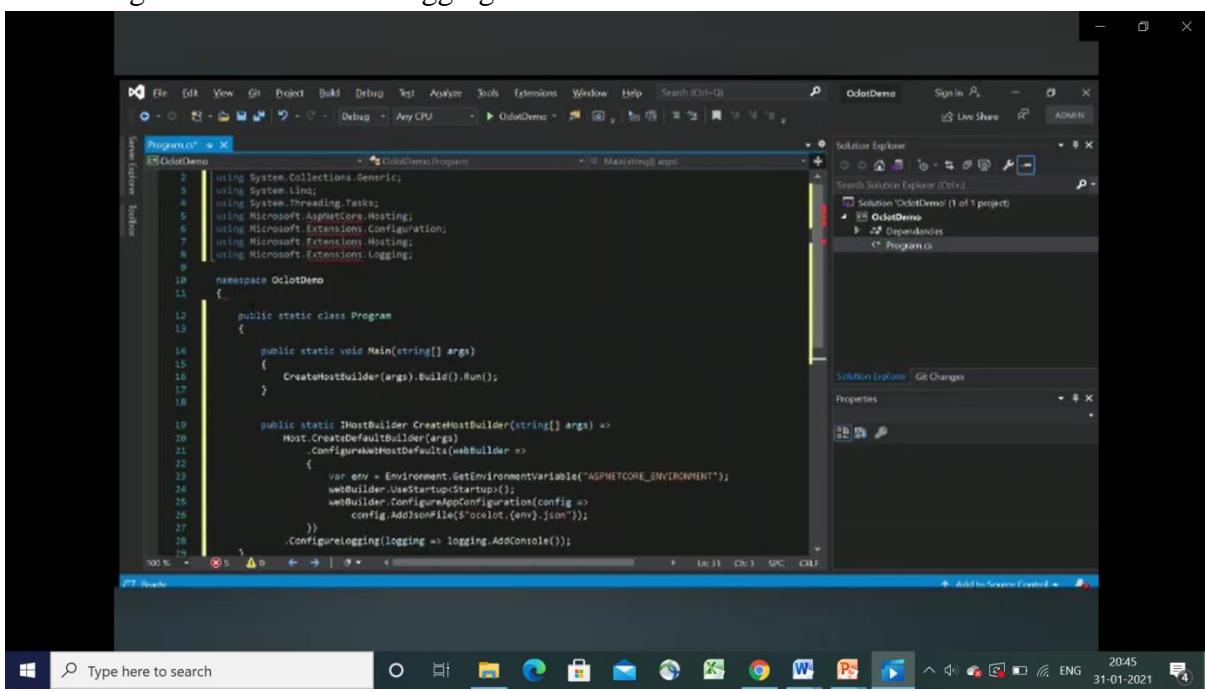
1. Create an ASP.NET Core Web Application Project.



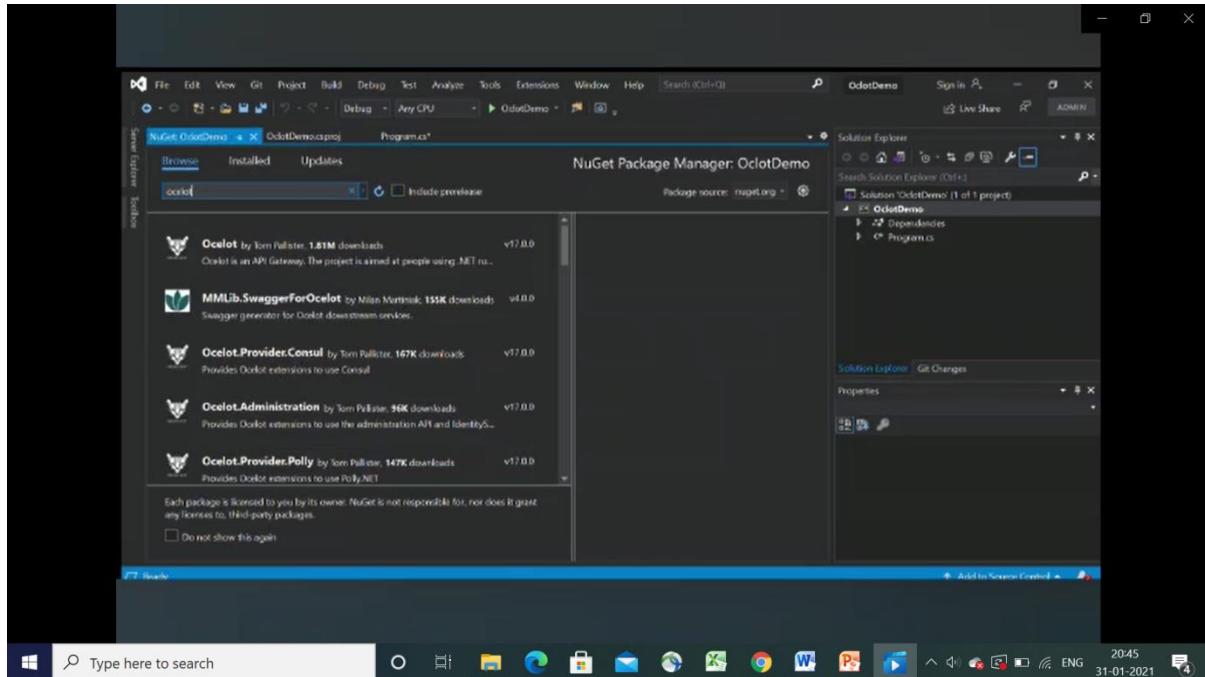
2. Create an empty ASP.NET Core 3.1 and give a name of the Project.



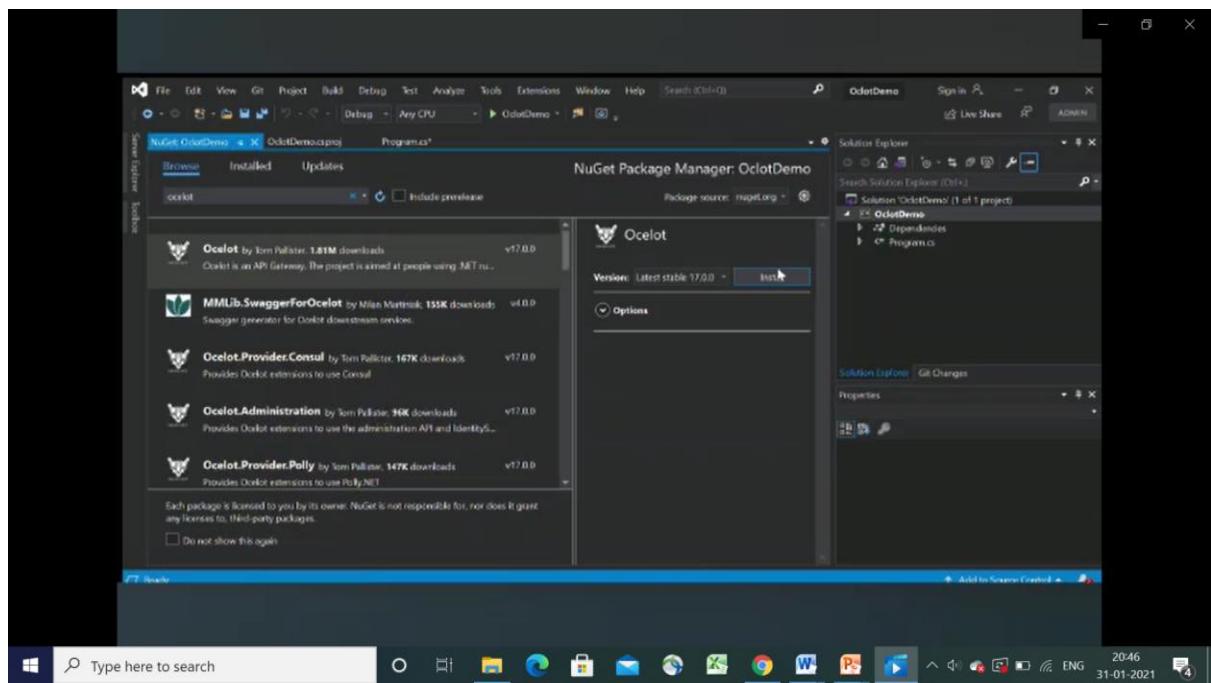
3. Go to Program.cs file and add logging code.



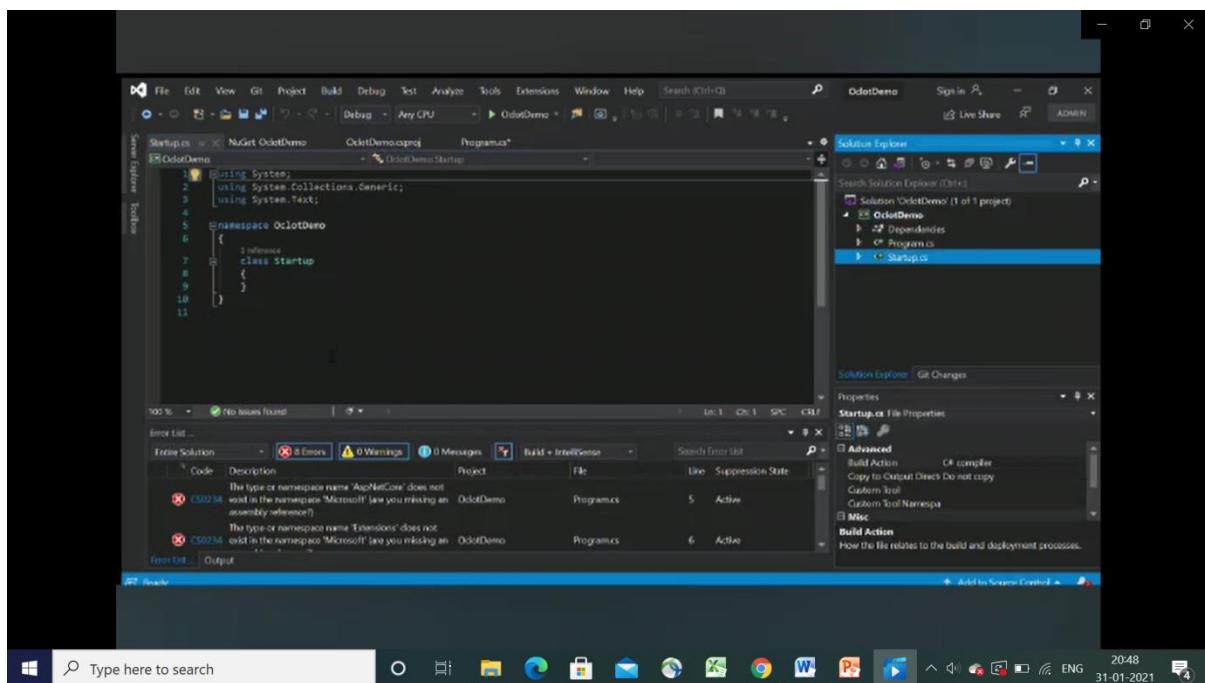
4. Now add new NuGet package for Ocelot



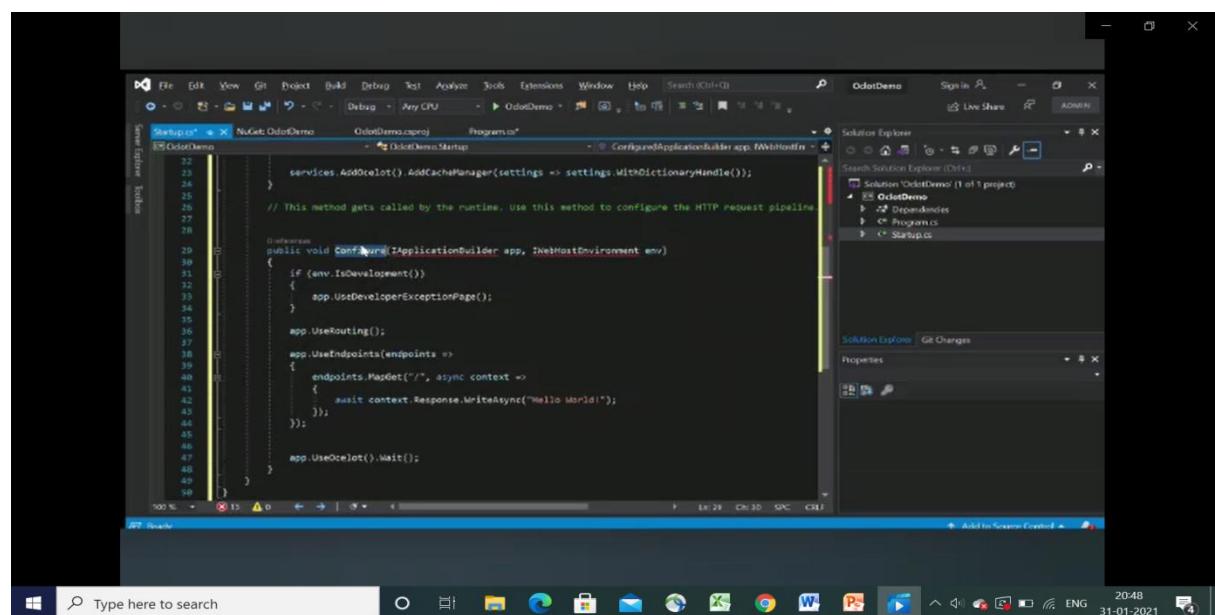
5. Click on Project -> Manage NuGet Package -> Click on Browse -> Search for ocelot package -> install.



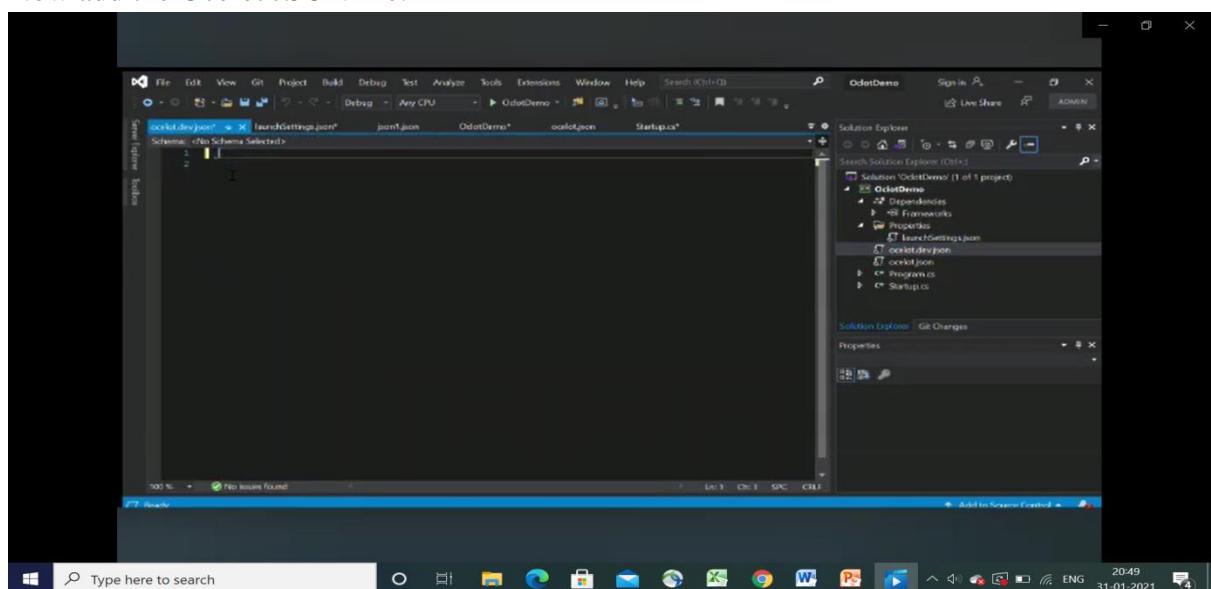
6. Once Ocelot Package is installed Go to Startup.cs



7. Now to Configure Ocelot add services.Addocelot() and app.UseOcelot().Wait() code.



8. Now add the Ocelot JSON file.

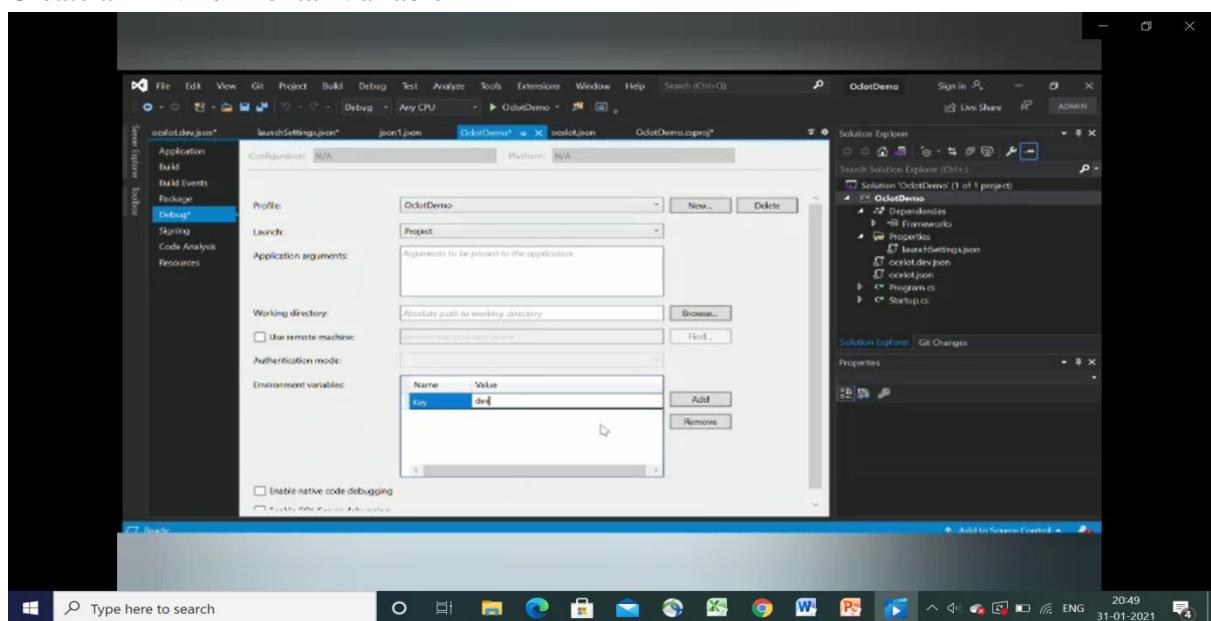


9. Click on Project name -> Add -> New Item -> JSON File ->(Give name)

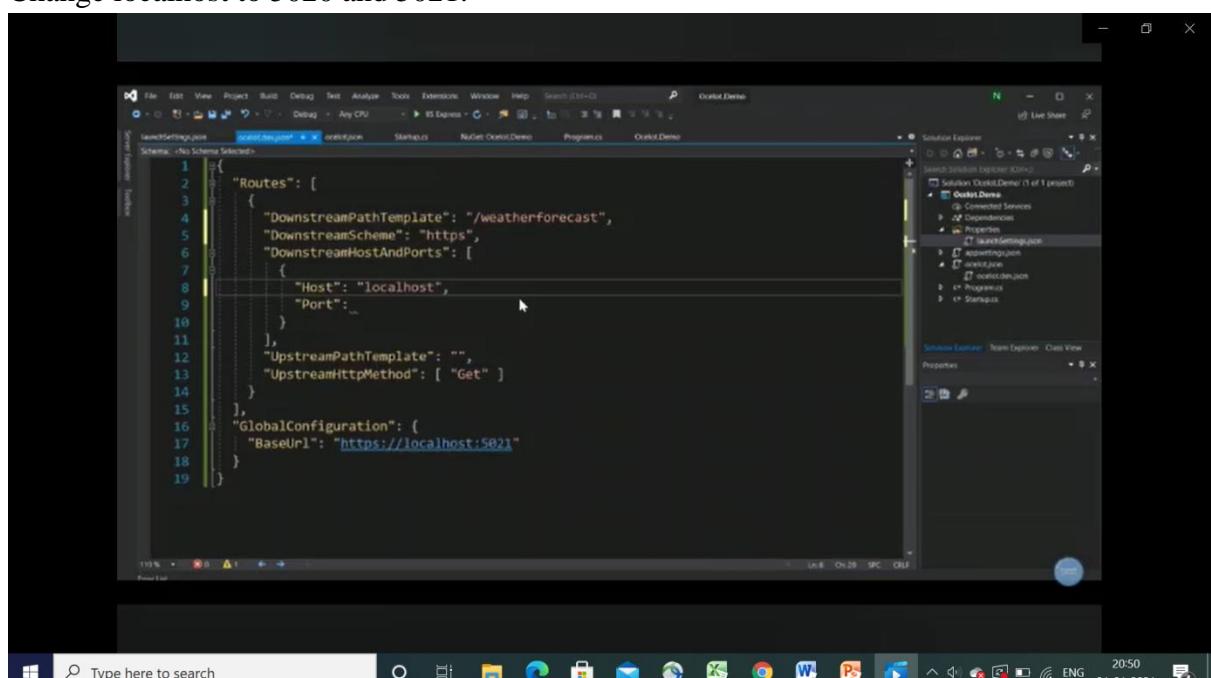
10. Add JSON Code.

```
1 "routes": [
2     {
3         "DownstreamPathTemplate": "/weatherforecast",
4         "DownstreamScheme": "http",
5         "DownstreamHostAndPorts": [
6             {
7                 "Host": "localhost",
8                 "Port": 5001
9             }
10            ],
11        "UpstreamPathTemplate": "/api/weather",
12        "UpstreamHttpMethod": [ "Get" ],
13        "AuthenticationOptions": [
14            {
15                "AuthenticationProviderKey": "Bearer",
16                "AllowedScopes": []
17            },
18            {
19                "RateLimitOptions": {
20                    "ClientWhitelist": [],
21                    "EnableRateLimiting": true,
22                    "Period": "5s",
23                    "PeriodInSpan": 1,
24                    "Limit": 1
25                },
26                "FileCacheOptions": { "TtlSeconds": 30 }
27            }
28        ],
29        "GlobalConfiguration": {
30            "BaseUrl": "https://localhost:5021"
31        }
32    }
33]
```

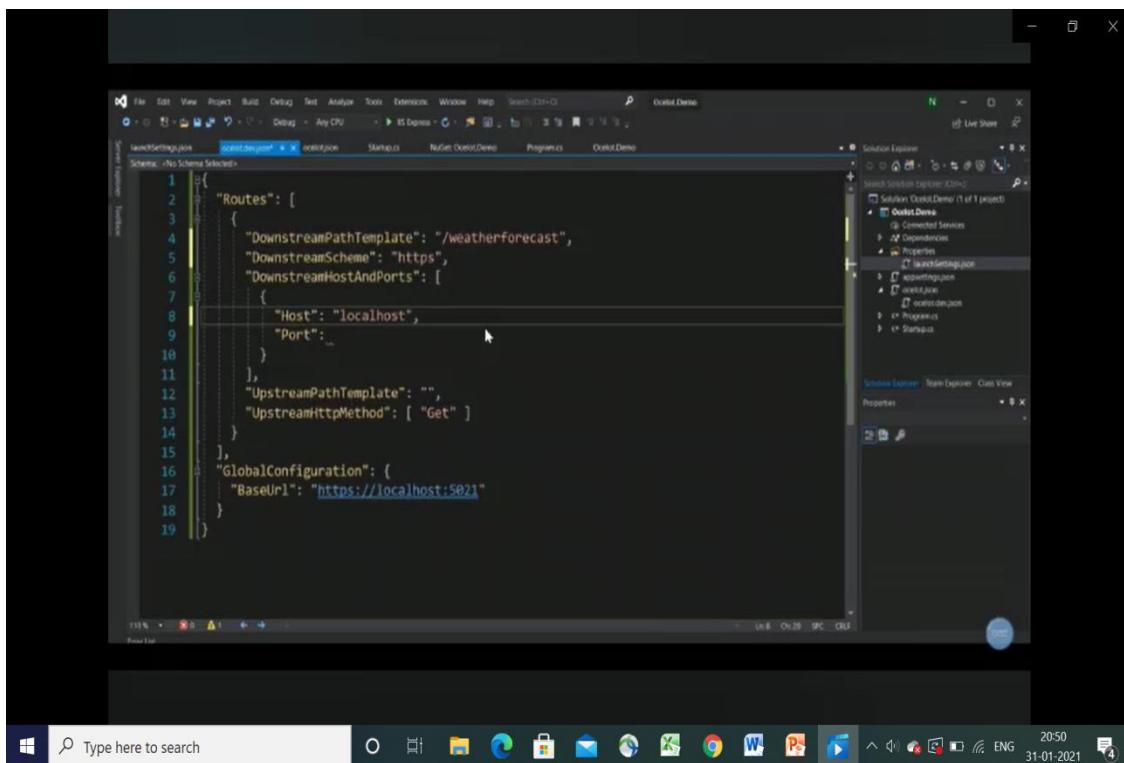
11. Create an Environmental Variable



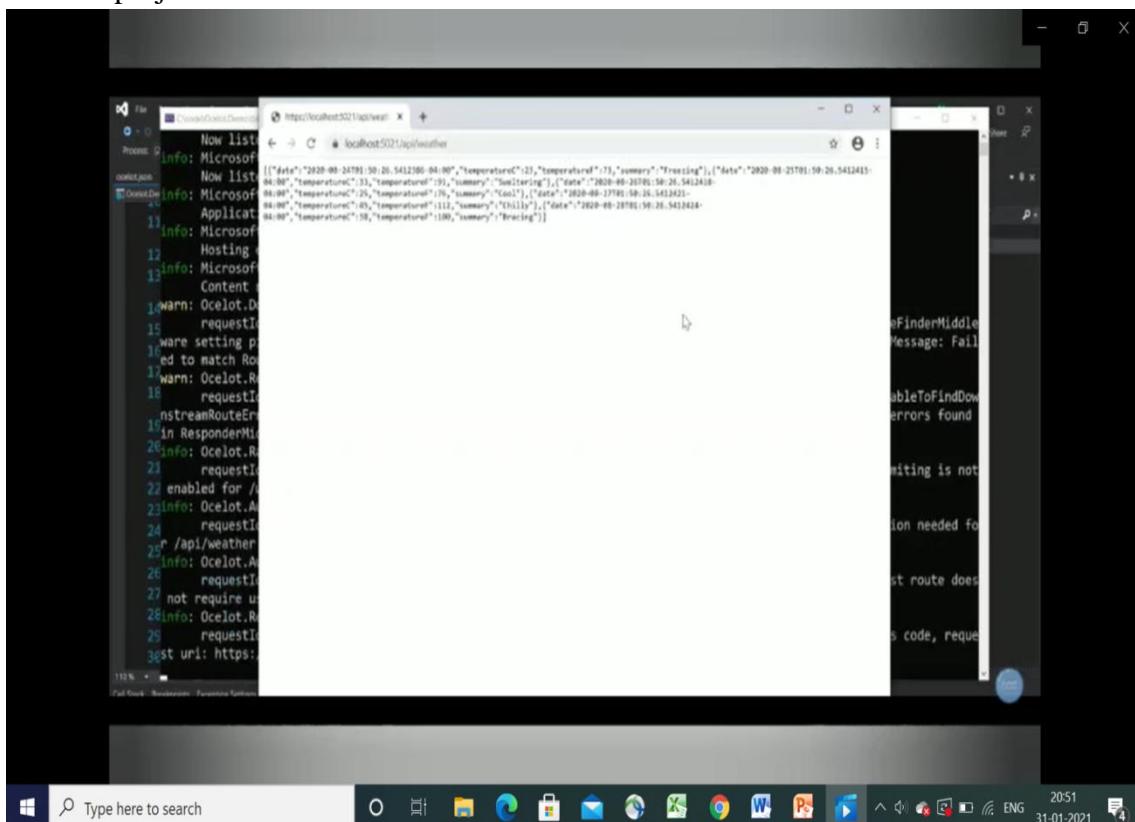
12. Change localhost to 5020 and 5021.



13. Add Configuration to the system.



14. Run the project.



Code:

Program.cs file

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Hosting;
```

```

using Microsoft.Extensions.Logging;
namespace Oclot.Demo
{
    public static class Program
    {
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().Run();
        }
        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    var env =
                        Environment.GetEnvironmentVariable("ASPNETCORE_ENVIRONMENT");
                    webBuilder.UseStartup<Startup>();
                    webBuilder.ConfigureAppConfiguration(config =>
                        config.AddJsonFile($"ocelot.{env}.json"));
                })
                .ConfigureLogging(logging => logging.AddConsole());
    }
}

```

Startup.cs file

```

namespace Oclot.Demo
{
    public class Startup
    {
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddOcelot().AddCacheManager(settings => settings.WithDictionaryHandle());
        }
        public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
        {
            if (env.IsDevelopment())
            {
                app.UseDeveloperExceptionPage();
            }
            app.UseRouting();
            app.UseEndpoints(endpoints =>
            {
                endpoints.MapGet("/", async context =>
                {
                    await context.Response.WriteAsync("Hello World!");
                });
            });
            app.UseOcelot().Wait();
        }
    }
}

```

JSON file

```
{
    "Routes": [
        {
            "DownstreamPathTemplate": "/weatherforecast",
            "DownstreamScheme": "https",
            "DownstreamHostAndPorts": [
                {
                    "Host": "localhost",
                    "Port": 5001
                }
            ],
            "UpstreamPathTemplate": "/api/weather",
            "UpstreamHttpMethod": [ "Get" ],
            "AuthenticationOptions": {
                "AuthenticationProviderKey": "Bearer",
                "AllowedScopes": []
            },
            "RateLimitOptions": {
                "ClientWhitelist": [],
                "EnableRateLimiting": true,
            }
        }
    ]
}
```

```

    "Period": "5s",
    "PeriodTimespan": 1,
    "Limit": 1 },
    "FileCacheOptions": { "TtlSeconds": 30 }
  ],
  "GlobalConfiguration": {
    "BaseUrl": "https://localhost:5021"
  }
}

```

OUTPUT

The screenshot displays two windows on a Windows desktop. The left window is a terminal session titled 'C:\code\Ocelot.Demo' showing log entries from Ocelot middleware. The right window is a browser window showing a 404 error page with the message 'Failed to find downstream route'.

```

[...] Now listening on: https://localhost:5021
[...] Now listening on: http://localhost:5020
[...] Application started. Press Ctrl+C to shut down.
[...] Hosting environment: dev
[...] Content root path: C:\code\Ocelot.Demo
[...] requestId: 0#02751KKSJU8:00000003, previousRequestId: no previous request id, message: DownstreamRouteFinderMiddleware setting pipeline errors. IDownstreamRouteFinder returned Error Code: UnableToFindDownstreamRouteError Message: Failed to match Route configuration for upstream path: /favicon.ico, verb: GET.
[...] requestId: 0#02751KKSJU8:00000003, previousRequestId: no previous request id, message: Error Code: UnableToFindDownstreamRouteError Message: Failed to match Route configuration for upstream path: /favicon.ico, verb: GET. errors found in ResponderMiddleware. Setting error response. For request path: /favicon.ico, request method: GET
[...] requestId: 0#02751KKSJU8:00000005, previousRequestId: no previous request id, message: ClientRateLimitMiddleware[0]
[...] requestId: 0#02751KKSJU8:00000005, previousRequestId: no previous request id, message: no previous request id, message: EndpointRateLimiting is not enabled for /weatherforecast
[...] requestId: 0#02751KKSJU8:00000005, previousRequestId: no previous request id, message: No authentication needed for /api/weather
[...] requestId: 0#02751KKSJU8:00000005, previousRequestId: no previous request id, message: /weatherforecast route does not require user to be authorised
[...] requestId: 0#02751KKSJU8:00000005, previousRequestId: no previous request id, message: 200 (OK) status code, request uri: https://localhost:5021/weatherforecast

```

The screenshot shows a single terminal window on a Windows desktop displaying Ocelot middleware logs for a weather API call. The logs show the middleware processing the request through various stages, including route finding and rate limiting.

```

[...] Now listening on: https://localhost:5021
[...] Now listening on: http://localhost:5020
[...] Application started. Press Ctrl+C to shut down.
[...] Hosting environment: dev
[...] Content root path: C:\code\Ocelot.Demo
[...] requestId: 0#02751KKSJU8:00000003, previousRequestId: no previous request id, message: DownstreamRouteFinderMiddleware setting pipeline errors. IDownstreamRouteFinder returned Error Code: UnableToFindDownstreamRouteError Message: Failed to match Route configuration for upstream path: /favicon.ico, verb: GET.
[...] requestId: 0#02751KKSJU8:00000003, previousRequestId: no previous request id, message: Error Code: UnableToFindDownstreamRouteError Message: Failed to match Route configuration for upstream path: /favicon.ico, verb: GET. errors found in ResponderMiddleware. Setting error response. For request path: /favicon.ico, request method: GET
[...] requestId: 0#02751KKSJU8:00000005, previousRequestId: no previous request id, message: ClientRateLimitMiddleware[0]
[...] requestId: 0#02751KKSJU8:00000005, previousRequestId: no previous request id, message: no previous request id, message: EndpointRateLimiting is not enabled for /weatherforecast
[...] requestId: 0#02751KKSJU8:00000005, previousRequestId: no previous request id, message: No authentication needed for /api/weather
[...] requestId: 0#02751KKSJU8:00000005, previousRequestId: no previous request id, message: /weatherforecast route does not require user to be authorised
[...] requestId: 0#02751KKSJU8:00000005, previousRequestId: no previous request id, message: 200 (OK) status code, request uri: https://localhost:5021/weatherforecast

```

Practical no 7

Create a database design for Microservices an application using the database.

Choosing a Data Store

There are many risks associated with embracing a 1.0-level technology. The ecosystem is generally immature, so support for your favorite things may be lacking or missing entirely. Tooling and integration and overall developer experience are often high-friction. Despite the long and storied history of

.NET, .NET Core (and especially the associated tooling) should still be treated like a brand new 1.0 product.

One of things we might run into when trying to pick a data store that is compatible with EF Core is a lack of available providers. While this list will likely have grown by the time you read this, at the time this chapter was written, the following providers were available for EF Core:

- SQL Server
- SQLite
- Postgres
- IBM databases
- MySQL
- SQL Server Lite
- In-memory provider for testing
- Oracle

For databases that aren't inherently compatible with the Entity Framework relational model, like MongoDB, Neo4j, Cassandra, etc., you should be able to find client libraries available that will work with .NET Core. Since most of these databases expose simple RESTful APIs, you should still be able to use them even if you have to write your own client.

Because of my desire to keep everything as cross-platform as possible throughout this book, I decided to use Postgres instead of SQL Server to accommodate readers working on Linux or Mac workstations. Postgres is also easily installed on Windows.

Building a Postgres Repository

In order to get something running and focus solely on the discipline and code required to stand up a simple service, we used an in-memory repository that didn't amount to much more than a fake that aided us in writing tests.

In this section we're going upgrade our location service to work with Postgres. To do this we're going to create a new repository implementation that encapsulates the PostgreSQL client communication. Before we get to the implementation code, let's revisit the interface for our location repository.

Example 1. ILocationRecordRepository.cs

```
using System;
using System.Collections.Generic;

namespace StatlerWaldorfCorp.LocationService.Models { public

    interface ILocationRecordRepository {
        LocationRecord Add(LocationRecord locationRecord); LocationRecord
        Update(LocationRecord locationRecord); LocationRecord Get(Guid
        memberId, Guid recordId); LocationRecord Delete(Guid memberId, Guid
        recordId);

        LocationRecord GetLatestForMember(Guid memberId);

        ICollection<LocationRecord> AllForMember(Guid memberId);
    }
}
```

The location repository exposes standard CRUD functions like Add, Update, Get, and Delete. In addition, this repository exposes methods to obtain the latest location entry for a member as well as the entire location history for a member.

The purpose of the location service is solely to track location data, so you'll notice that there is no reference to team membership at all in this interface.

Creating a Database Context

The next thing we're going to do is create a database context. This class will serve as a wrapper around the base DbContext class we get from Entity Framework Core. Since we're dealing with locations, we'll call our context class LocationDbContext.

If you're not familiar with Entity Framework or EF Core, the database context acts as the gateway between your database-agnostic model class (POCOs, or Plain-Old C# Objects) and the real database. For more information on EF Core, check out Microsoft's documentation. We could probably spend another several chapters doing nothing but exploring its details, but since we're trying to stay focused on cloud-native applications and services, we'll use just enough EF Core to build our services.

The pattern for using a database context is to create a class that inherits from it that is specific to your model. In our case, since we're dealing with locations, we'll create a LocationDbContext class.

Example 2. LocationDbContext.cs

```
using Microsoft.EntityFrameworkCore;
using StatlerWaldorfCorp.LocationService.Models; using
Npgsql.EntityFrameworkCore.PostgreSQL;

namespace StatlerWaldorfCorp.LocationService.Persistence
{
    public class LocationDbContext : DbContext
    {
        public LocationDbContext(
            DbContextOptions<LocationDbContext> options) :
            base(options)
        {
        }

        protected override void OnModelCreating(
            ModelBuilder modelBuilder)
        {
            base.OnModelCreating(modelBuilder);
            modelBuilder.HasPostgresExtension("uuid-ossp");
        }

        public DbSet<LocationRecord> LocationRecords {get; set;}
    }
}
```

Here we can use the ModelBuilder and DbContextOptions classes to perform any additional setup we need on the context. In our case, we're ensuring that our model has the `uuid-osspPostgres` extension to support the member ID field.

Implementing the Location Record Repository Interface

Now that we have a context through which other classes can use to communicate with the database, we can create a real implementation of the ILocationRecordRepository interface. This real implementation will take an instance of LocationDbContext as a constructor parameter. This sets us up nicely to configure this context with environment-supplied connection strings when deploying for real and with mocks or in-memory providers when testing.

Example 3 contains the code for the LocationRecordRepository class

Example 3. LocationRecordRepository.cs

```
using System; using
System.Linq;
using System.Collections.Generic;
using StatlerWaldorfCorp.LocationService.Models;

namespace StatlerWaldorfCorp.LocationService.Persistence
{
    public class LocationRecordRepository :
        ILocationRecordRepository
    {
        private LocationDbContext context;

        public LocationRecordRepository(LocationDbContext context)
        {
            this.context = context;
        }

        public LocationRecord Add(LocationRecord locationRecord)
        {
            this.context.Add(locationRecord);
            this.context.SaveChanges(); return
            locationRecord;
        }

        public LocationRecord Update(LocationRecord locationRecord)
        {
            this.context.Entry(locationRecord).State =
                EntityState.Modified;
            this.context.SaveChanges(); return
            locationRecord;
        }

        public LocationRecord Get(Guid memberId, Guid recordId)
        {
            return this.context.LocationRecords
                .Single(lr => lr.MemberID == memberId &&
                lr.ID == recordId);
        }

        public LocationRecord Delete(Guid memberId, Guid recordId)
        {
            LocationRecord locationRecord =
                this.Get(memberId, recordId);
            this.context.Remove(locationRecord);
            this.context.SaveChanges();
            return locationRecord;
        }
    }
}
```

```

    }

    public LocationRecord GetLatestForMember(Guid memberId)
    {
        LocationRecord locationRecord =
            this.context.LocationRecords.
                Where(lr => lr.MemberID == memberId).
                OrderBy(lr => lr.Timestamp).
                Last();
        return locationRecord;
    }

    public ICollection<LocationRecord> AllForMember(Guid memberId)
    {
        return this.context.LocationRecords.
            Where(lr => lr.MemberID == memberId).
            OrderBy(lr => lr.Timestamp).
            ToList();
    }
}

```

The code here is pretty straightforward. Any time we make a change to the database, we call `SaveChanges` on the context. If we need to query, we use the LINQ expression syntax where we can combine `Where` and `OrderBy` to filter and sort the results.

When we do an update, we need to flag the entity we're updating as a modified entry so that Entity Framework Core knows how to generate an appropriate SQL UPDATE statement for that record. If we don't modify this entry state, EF Core won't know anything has changed and so a call to `SaveChanges` will do nothing.

The next big trick in this repository is injecting the Postgres-specific database context. To make this happen, we need to add this repository to the dependency injection system in the `ConfigureServices` method of our `Startup` class.

Example 4. `ConfigureServices` method in `Startup.cs`

```

public void ConfigureServices(IServiceCollection services)
{
    services.AddEntityFrameworkNpgsql()
        .AddDbContext<LocationDbContext>(options =>
            options.UseNpgsql(Configuration));
    services.AddScoped<ILocationRecordRepository, LocationRecordRepository>();
    services.AddMvc();
}

```

First we want to use the `AddEntityFrameworkNpgsql` extension method exposed by the Postgres EF Core provider. Next, we add our location repository as a scoped service. When we use the `AddScoped` method, we're indicating that every new request made to our service gets a newly created instance of this repository.

Configuring a Postgres Database Context

The repository we built earlier requires some kind of database context in order to function. The database context is the core primitive of Entity Framework Core. To create a database context for the location model, we just need to create a class that inherits from `DbContext`. I've also included a `DbContextFactory` because that can sometimes make running the Entity Framework Core command-line tools simpler:

```

        optionsBuilder.UseNpgsql(connectionString);

        return new LocationDbContext(optionsBuilder.Options);
    }
}
}
}

```

With a new database context, we need to make it available for dependency injection so that the location repository can utilize it:

```

public void ConfigureServices(IServiceCollection services)
{
    var transient = true;
    if (Configuration.GetSection("transient") != null) { transient =
        Boolean.Parse(Configuration
            .GetSection("transient").Value);
    }
    if (transient) { logger.LogInformation(
        "Using transient location record repository.");
        services.AddScoped<ILocationRecordRepository,
            InMemoryLocationRecordRepository>();
    } else {
        var connectionString = Configuration.GetSection("postgres:cstr").Value;

        services.AddEntityFrameworkNpgsql()
            .AddDbContext<LocationDbContext>(options =>
                options.UseNpgsql(connectionString));
        logger.LogInformation(
            "Using '{0}' for DB connection string.",
            connectionString);
        services.AddScoped<ILocationRecordRepository,
            LocationRecordRepository>();
    }

    services.AddMvc();
}

```

The calls to `AddEntityFrameworkNpgsql` and `AddDbContext` are the magic that makes everything happen here.

With a context configured for DI, our service should be ready to run, test, and accept EF Core command-line parameters like the ones we need to execute migrations. When building your own database-backed services, you can also use the EF Core command-line tools to reverse-engineer migrations from existing database schemas.

Exercising the Data Service

Running the data service should be relatively easy. The first thing we're going to need to do is spin up a running instance of Postgres. If you were paying attention to the `wercker.yml` file for the location service that sets up the integration tests, then you might be able to guess at the `docker run` command to start Postgres with our preferred parameters:

```
$ docker run -p 5432:5432 --name some-postgres \
-e POSTGRES_PASSWORD=inteword -e POSTGRES_USER=integrator \
-e POSTGRES_DB=locationservice -d postgres
```

This starts the Postgres Docker image with the name `some-postgres` (this will be important shortly). To verify that we can connect to Postgres, we can run the following Docker command to launch `psql`:

```
$ docker run -it --rm --link some-postgres:postgres postgres \
psql -h postgres -U integrator -d locationservice
```

With the database up and running, we need a schema. The tables in which we expect to store the migration metadata and our location records don't yet exist. To put them in the database, we just need to run an EF Core command from the location service's project directory. Note that we're also setting environment variables that we'll need soon:

```
$ export TRANSIENT=false
$ export POSTGRES_CSTR="Host=localhost;Username=integrator; \
Password=inteword;Database=locationservice;Port=5432"
$ dotnet ef database update
```

At this point Postgres is running with a valid schema and it's ready to start accepting commands from the location service. Here's where it gets a little tricky. If we're going to run the location service from inside a Docker image, then referring to the Postgres server's host as `localhost` won't work —because that's the host inside the Docker image. What we need is for the location service to reach out of its container and then into the Postgres container. We can do this with a container link that creates a virtual hostname (we'll call it `postgres`), but we'll need to change our environment variable before launching the Docker image:

```
$ export POSTGRES_CSTR="Host=postgres;Username=integrator; \
Password=inteword;Database=locationservice;Port=5432"
$ docker run -p 5000:5000 --link some-postgres:postgres \
-e TRANSIENT=false -e PORT=5000 \
-e POSTGRES_CSTR dotnetcoreservices/locationservice:latest
```

Now that we've linked the service's container to the Postgres container via the `postgres` hostname, the location service should have no trouble connecting to the database. To see this all in action, let's submit a location record (as usual, take the line feeds out of this command when you type it):

```
$ curl -H "Content-Type:application/json" -X POST -d \
'{"id":"64c3e69f-1580-4b2f-a9ff-2c5f3b8f0e1f","latitude":12.0, \
"longitude":10.0,"altitude":5.0,"timestamp":0, \
"memberId":"63e7acf8-8fae-42ce-9349-3c8593ac8292"}' \
http://localhost:5000/locations/63e7acf8-8fae-42ce-9349-3c8593ac8292
```

Take a look at the trace output from your running Docker image for the location service. You should see some very useful Entity Framework trace data explaining what happened. The service performed a SQL INSERT, so things are looking promising:

```
info: Microsoft.EntityFrameworkCore.Storage.

IRelationalCommandBuilderFactory[1]
    Executed DbCommand (23ms)
[Parameters=[@p0='?', @p1='?', @p2='?', @p3='?', @p4='?', @p5='?'],
 CommandType='Text', CommandTimeout='30']
    INSERT INTO "LocationRecords" ("ID", "Altitude", "Latitude",
"Longitude", "MemberID", "Timestamp")
    VALUES (@p0, @p1, @p2, @p3, @p4, @p5);
info: Microsoft.AspNetCore.Mvc.Internal.ObjectResultExecutor[1]
    Executing ObjectResult, writing value Microsoft.AspNetCore
.Mvc.ControllerContext.
info: Microsoft.AspNetCore.Mvc.Internal.ControllerActionInvoker[2]
    Executed action StatlerWaldorfCorp.LocationService.
Controllers.LocationRecordController.AddLocation
(StatlerWaldorfCorp.LocationService) in 2253.7616ms
info: Microsoft.AspNetCore.Hosting.Internal.WebHost[2]
    Request finished in 2602.7855ms 201 application/json;
charset=utf-8
```

Let's ask the service for this fictitious member's location history:

```
$ curl http://localhost:5000/locations/63e7acf8-8fae-42ce-9349-
3c8593ac8292

[{"id":"64c3e69f-1580-4b2f-a9ff-2c5f3b8f0e1f",
"latitude":12.0,"longitude":10.0,"altitude":5.0,
"timestamp":0,"memberID":"63e7acf8-8fae-42ce-9349-3c8593ac8292"}]
```

The corresponding Entity Framework trace looks like this:

```
info: Microsoft.EntityFrameworkCore.Storage.  
IRelationalCommandBuilderFactory[1]  
      Executed DbCommand (23ms) [Parameters=@__memberId_0='?'],  
      CommandType='Text', CommandTimeout='30'  
      SELECT "lr"."ID", "lr"."Altitude", "lr"."Latitude",  
      "lr"."Longitude", "lr"."MemberID", "lr"."Timestamp"  
      FROM "LocationRecords" AS "lr"  
      WHERE "lr"."MemberID" = @__memberId_0  
      ORDER BY "lr"."Timestamp"
```

Just to be double sure, let's query the latest endpoint to make sure we still get what we expect to see:

```
$ curl http://localhost:5000/locations/63e7acf8-8fae-42ce-9349-  
3c8593ac8292 \  
/latest  
  
{"id": "64c3e69f-1580-4b2f-a9ff-2c5f3b8f0e1f",  
 "latitude": 12.0, "longitude": 10.0, "altitude": 5.0,  
 "timestamp": 0, "memberID": "63e7acf8-8fae-42ce-9349-3c8593ac8292"}
```

Finally, to prove that we really are using real database persistence and that this isn't just a random fluke, use docker ps and docker kill to locate the Docker process for the location service and kill it. Restart it using the exact same command you used before. You should now be able to query the location service and get the exact same data you had before. Of course, once you stop the Postgres container you'll permanently lose that data.

PRACTICAL 8

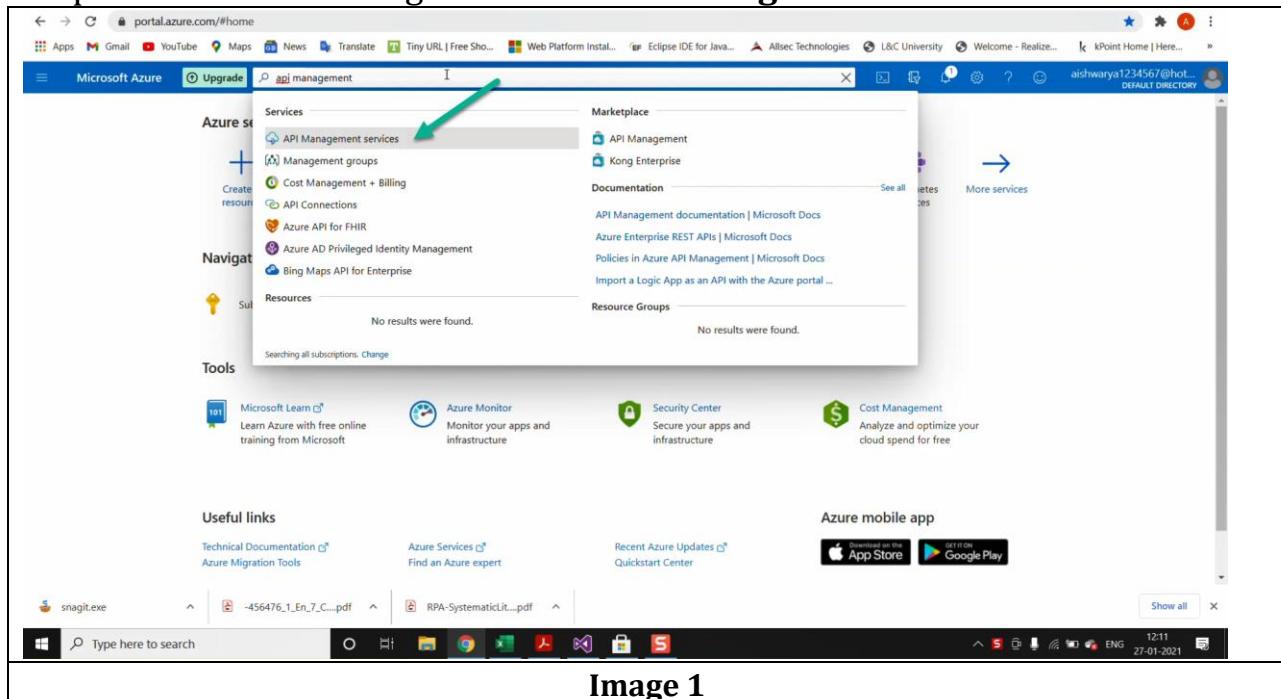
Create an API gateway service

a) Create an API Management Service.
Steps for creating an API management service are mentioned below-

Step 1- Sign-in to your **Azure Subscription Portal**

Step 2- Search for “**API Management**”, then select API Management in order to create a service instance.

Step 3- As shown in Image 1 Select “**API Management service**”



Step 4- As API management service page is loaded, select “**Create API management service**” button. As shown in the image 2.

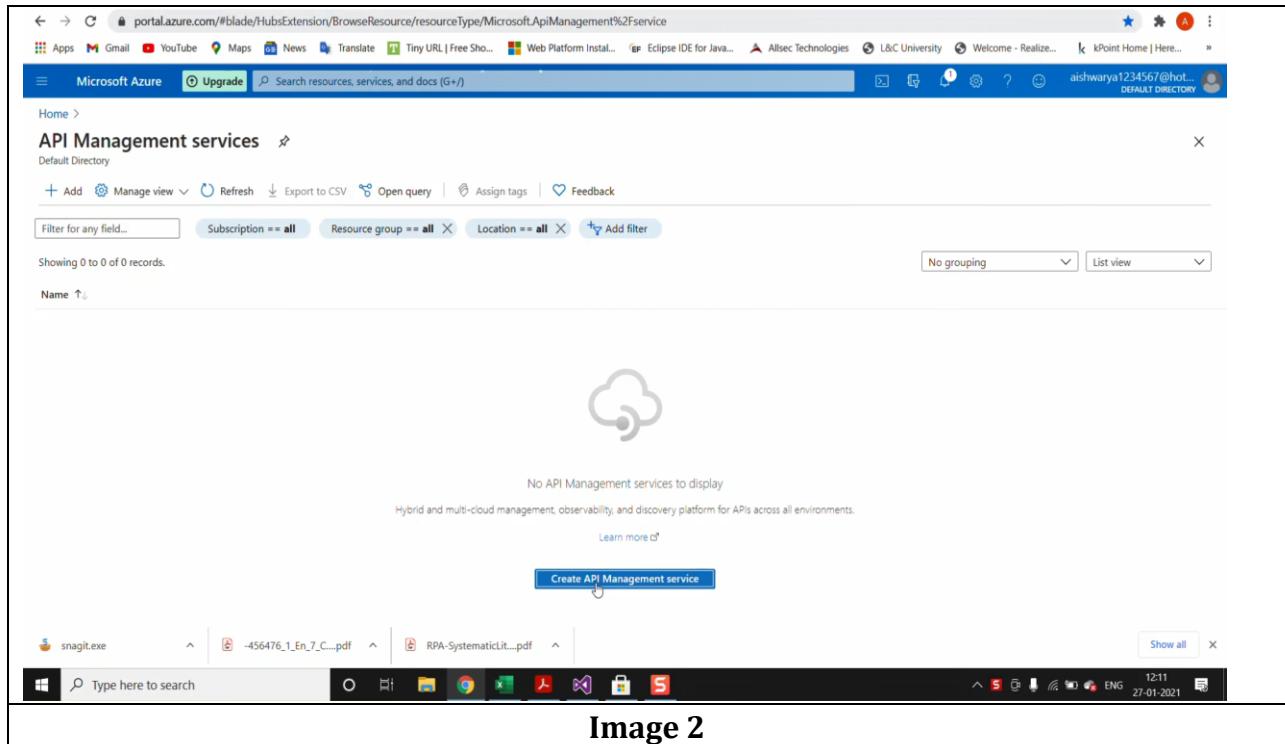


Image 2

Step 5- We will be able to see the Azure API Management service creation blade as shown in Image after clicking on the “Create API management service” button

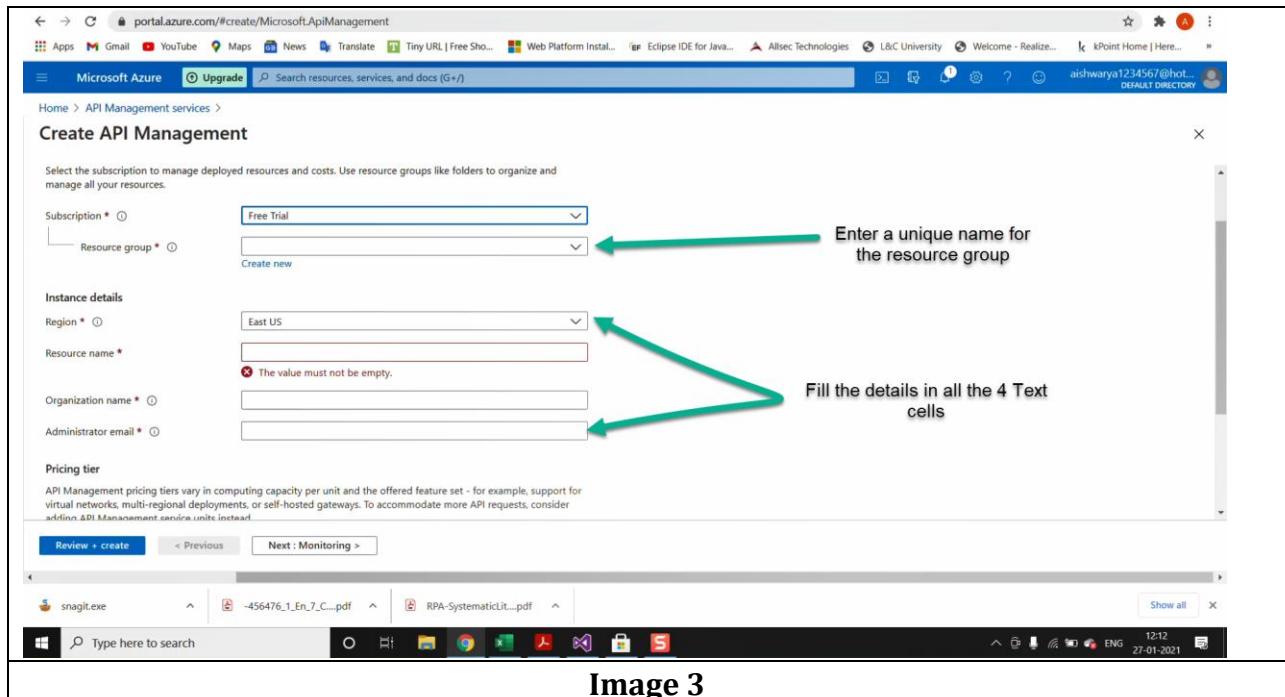


Image 3

Step 6- Provide a name. This name sets the URL of the API gateway and portal [Name- AzureManagementService]

Step 7-Select the subscription and resource group (or create a new resource

group), and select the location. [**Resource Name- AzureManagementService, Location- SouthEast Asia**]

Step 8-Specify the **organization name**. This name will appear in the developer portal as the organization that publishes the API. [**Name- CloudApplicationPractical**]

Step 9- Specify the **email address** of the administrator. The user who creates the service instance will be the default administrator, so it's best to provide the email address of this user until you want someone else to serve as administrator. [**Email ID- aishwarya1234567@hotmail.com**]

Step 10 -Select the pricing tier. The **Developer tier** is the most comprehensive offering, with sufficient request/response limitations in dev/test scenarios.

Step 11-After Completing the form click on **review + create** button to create “Azure API Management service instance”. As shown in image 4

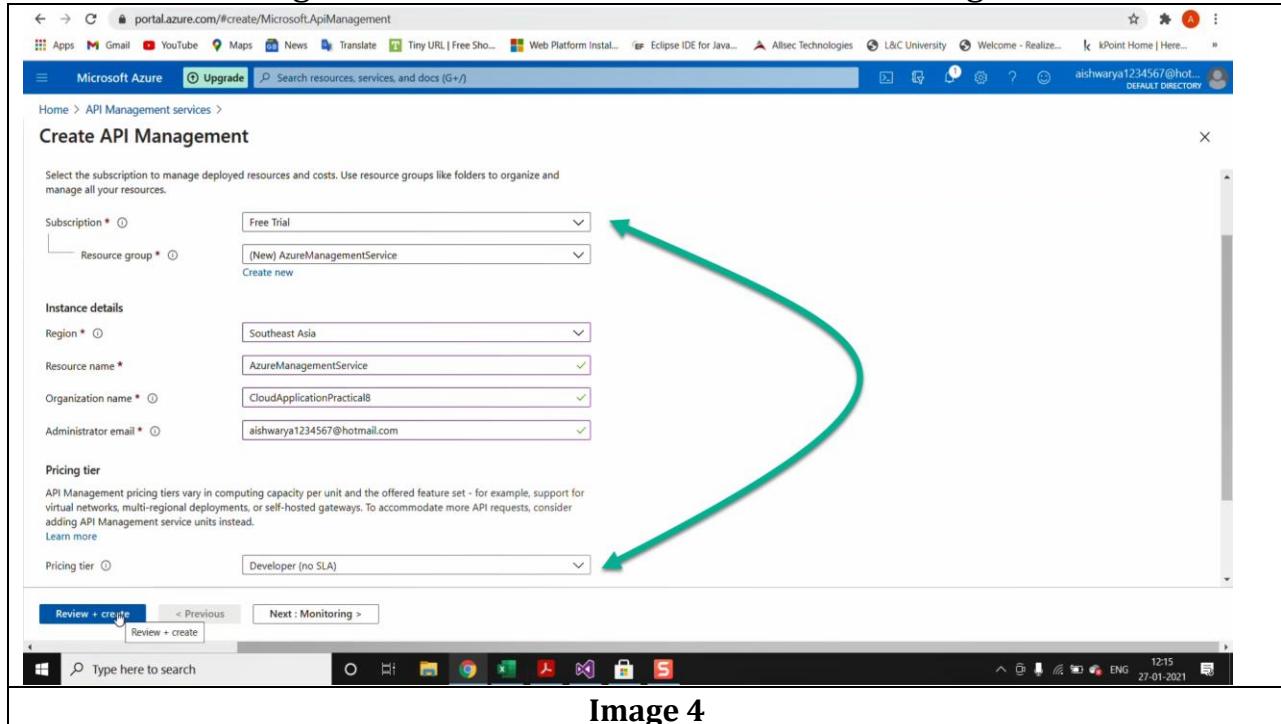


Image 4

Step 12-As shown in the image 5 the status is “**Running Final Validation**”. Wait for it to be in “**Validation Passed**” Stage.

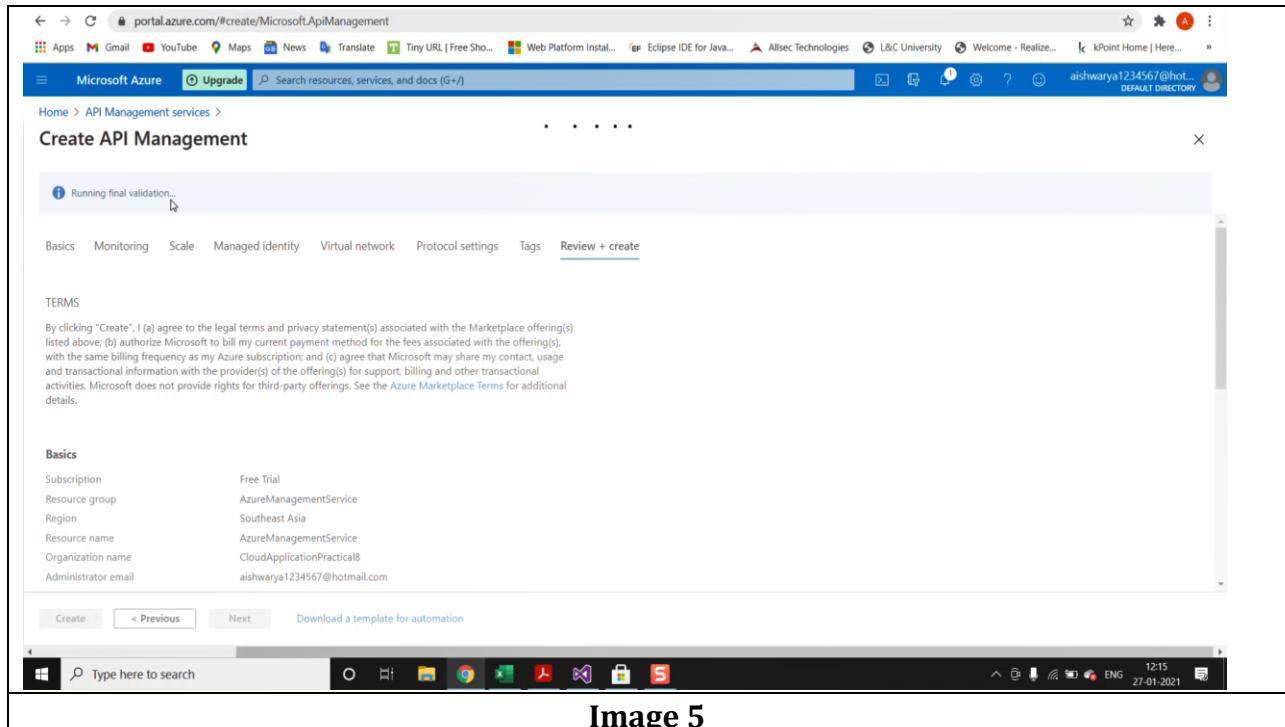


Image 5

Step 13-Once the Validation is done and the label displays “Validation Passed”, Click on **create** button to create the **instance of the service**.

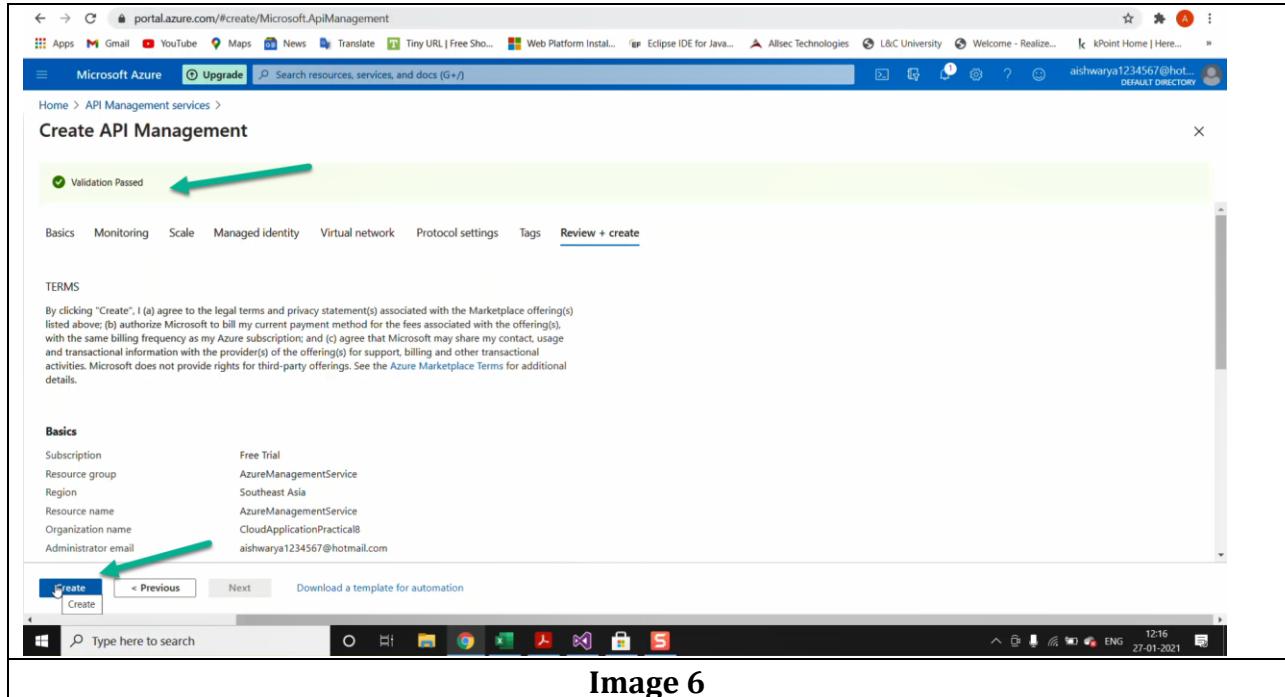


Image 6

Step 14- After clicking on the create button the API management service instance will go to deployment stage as shown in image 7. After the

deployment is complete the instance will be shown as in image 8

The screenshot shows the Microsoft Azure portal interface. At the top, there's a search bar and a user profile. Below it, the title is "Microsoft.ApiManagement-20210127121144 | Overview". A navigation menu on the left includes "Overview", "Inputs", "Outputs", and "Template". The main content area displays a deployment status: "Deployment is in progress". It shows details like Deployment name: Microsoft.ApiManagement-20210127121144, Start time: 1/27/2021, 12:16:22 PM, Subscription: Free Trial, and Resource group: AzureManagementService. A table titled "Deployment details" lists one resource: AzureManagementService, Type: Microsoft.ApiManagement/service, Status: Created, and Operation details. On the right, there's a sidebar with links to Security Center, Free Microsoft tutorials, and Work with an expert. The taskbar at the bottom shows various pinned icons.

Image 7

This screenshot is identical to Image 7, showing the same deployment details and sidebar. However, it includes two green arrows. One arrow points from the text "Your deployment is complete" to the checkmark icon. Another arrow points from the "AzureManagementService" row in the "Deployment details" table to the "Go to resource" button below it. The taskbar at the bottom is also visible.

Image 8

Next step is Import an API into API Management and test the API in the Azure portal

Step 15- In the Azure portal, search for and select API Management services.
Step 16- On the API Management services page, select your API Management instance as in image 9.

The screenshot shows the Microsoft Azure portal with the URL [https://portal.azure.com/#@aishwarya1234567@hotmail.onmicrosoft.com/resource/subscriptions/e6c681a8-936a-441b-976e-d0d5b7b8ab2d/resourceGroups/AzureManagementService/providers/Microsoft.ApiManagement/apiManagementServices/AzureManagementService](#). The page title is "API Management services". The main content area displays a table with one record:

| Name | Status | Tier | Type | Location | Resource group | Subscription |
|------------------------|------------|-----------|------------------------|----------------|------------------------|--------------|
| AzureManagementService | Activating | Developer | API Management service | Southeast Asia | AzureManagementService | Free Trial |

Image 9

Step 17- In the left navigation of **API Management instance**, select APIs.
Step 18- Select the **OpenAPI** tile as shown in image 10.

The screenshot shows the Microsoft Azure portal with the URL [https://apimanagement.hosting.portal.azure.net/apiManagement/Content/1.22.1.0.7/apimap/apimap-apis/index.html?clientOptimizations=undefined&l=en-us&trustedAuthority=https%3A%2F%2Fportal.azure.com&shellVersion=undefined#](#). The page title is "AzureManagementService | APIs". The left sidebar shows a navigation tree with "APIs" selected. The main content area shows a grid of API types:

| | | |
|-----------|-------------|--------------|
| Blank API | OpenAPI | WADL |
| Logic App | App Service | Function App |

Image 10

Step 19- In the Create from **OpenAPI** specification window, select Full.
Step 20- Enter the values as mentioned below in the form shown in image 11

| Setting | Value |
|-----------------------|---|
| OpenAPI specification | https://conferenceapi.azurewebsites.net?format=json |
| Display name | After you enter the preceding service URL, API Management fills out this field based on the JSON. .(automatic) |

| | |
|-----------------------|--|
| Name | After you enter the preceding service URL, API Management fills out this field based on the JSON. .(automatic) |
| Description | After you enter the preceding service URL, API Management fills out this field based on the JSON.(automatic) |
| URL scheme | HTTPS |
| API URL suffix | conference |
| Tags | -leave it as blank |
| Products | Unlimited |
| Gateways | Managed |

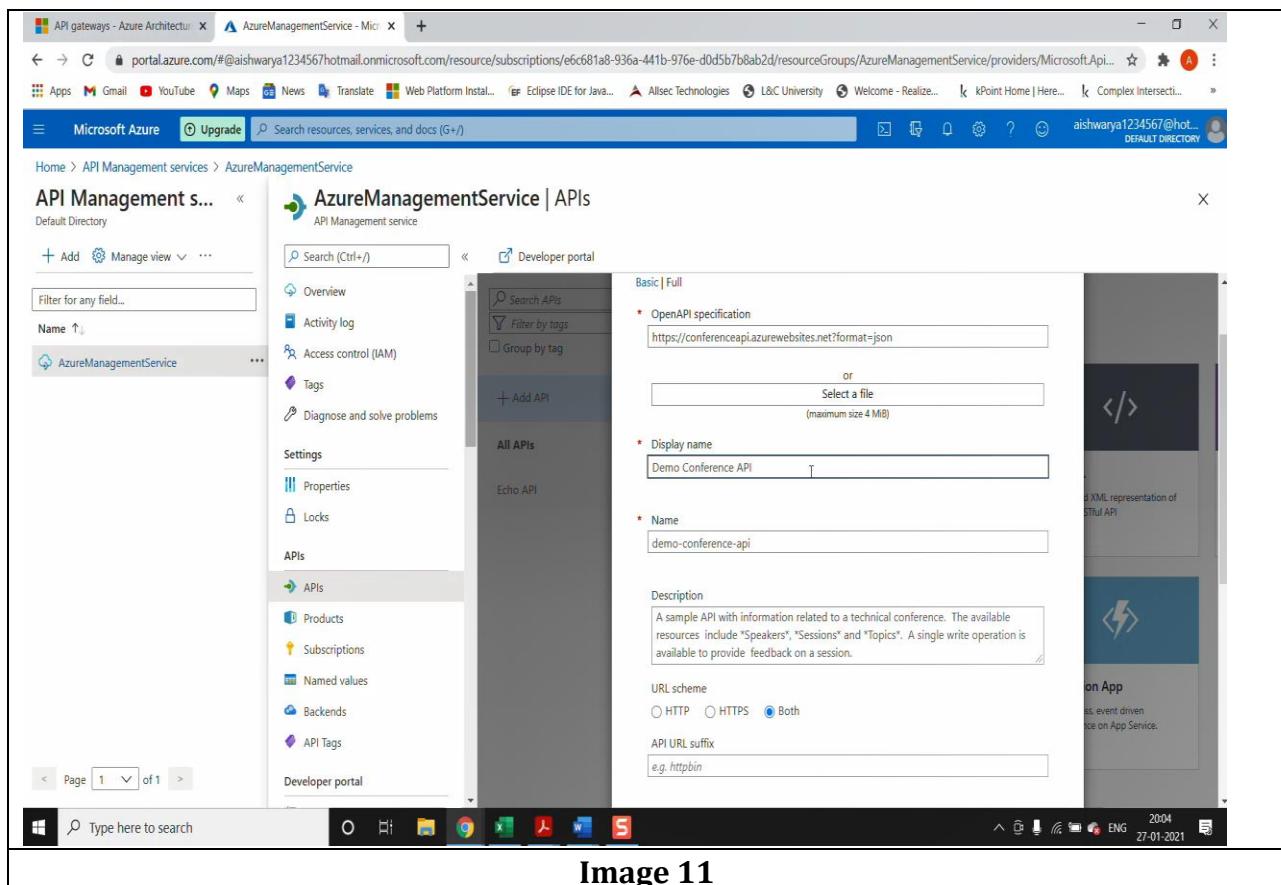


Image 11

Next steps are to Test the new API in the Azure portal

Step 21- In the left navigation of your API Management instance, select APIs > **Demo Conference API**.

Step 22- Select the Test tab, and then select **GetSpeakers**. The page shows Query parameters and Headers, if any. The Ocp-Apim-Subscription-Key is filled in automatically for the subscription key associated with this API.

Step 22- Select **Send** as shown in image 12.

Output -

The screenshot shows the Azure Management Service interface for the AzureManagementService. On the left, the 'APIs' section is selected under 'Settings'. In the center, the 'AzureManagementService | APIs' developer portal is displayed. A specific API, 'Demo Conference API', is selected. The 'Test' tab is active in the top navigation bar. Below it, a list of API operations is shown, with 'GetSpeakers' highlighted. To the right, the 'HTTP response' pane shows a successful '200 OK' response with detailed headers and a JSON payload. A green arrow labeled 'step 22' points to the 'Test' tab, another arrow labeled 'Step 22' points to the 'GetSpeakers' operation, and a third arrow labeled 'step 23' points to the 'Send' button.

Image 12

As we can see in image 12 the Https response is “200 Ok with some data” which states that the backend can response.

b) Create an API Gateway Service.

Steps for creating an API gateway service are mentioned below-

Step 1- Sign in to you AWS account.

Step 2- search for **API Gateway**, and create an AWS Gateway instance as shown in Image 1.

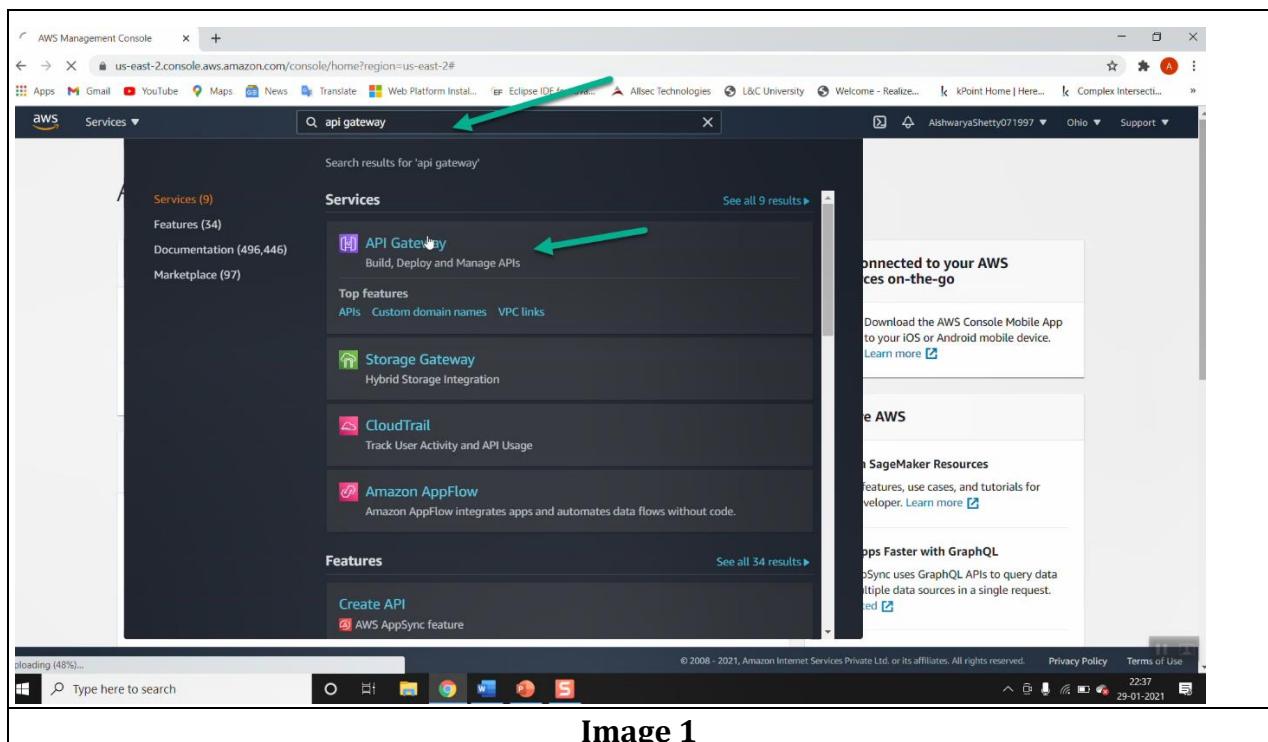


Image 1

Step 3- Select Import from the REST API selection as shown in image 2.

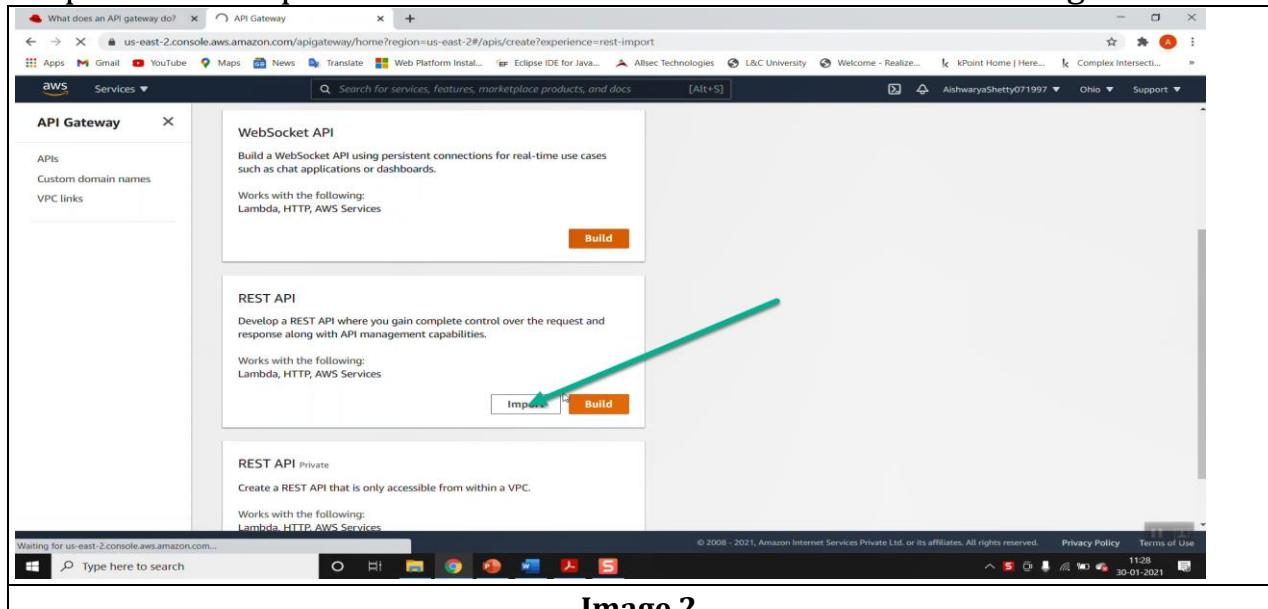


Image 2

Step 4- Select Import from swagger and copy or select the swagger file as shown in image 3.

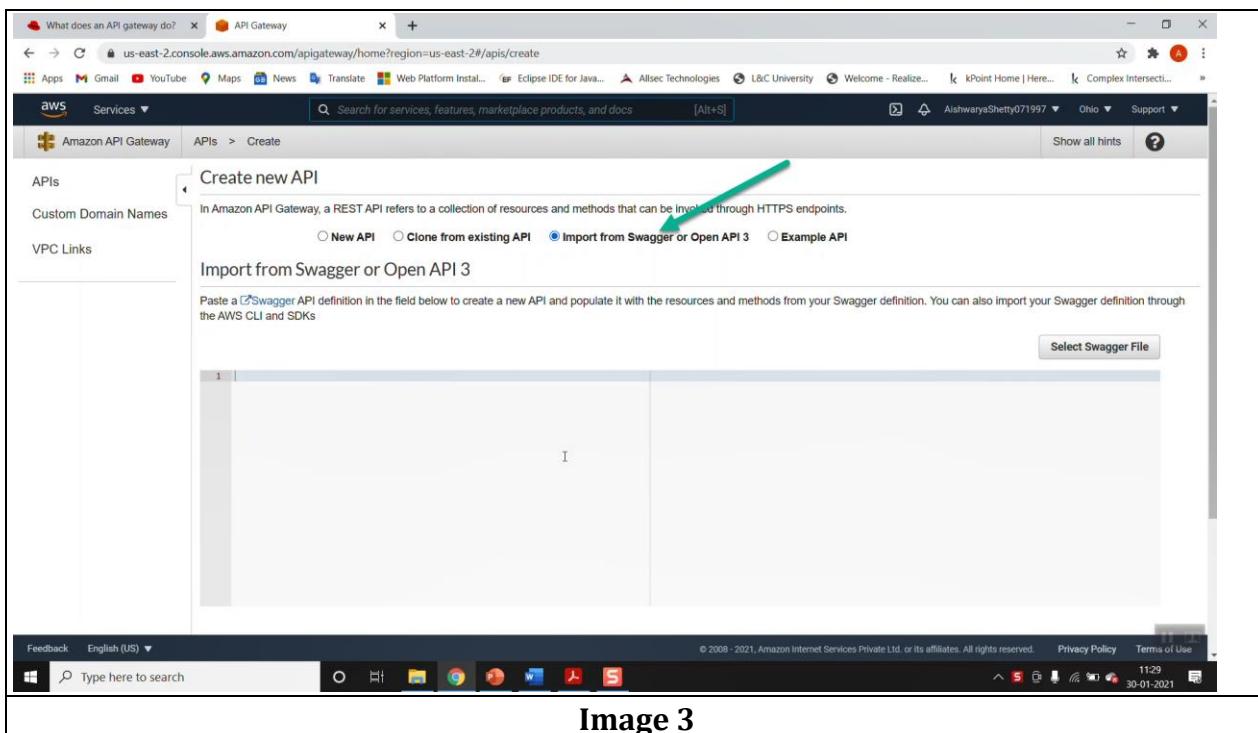


Image 3

Step 5- To Create an API definition in Swagger. First login to the swagger account. As depicted in image 4.

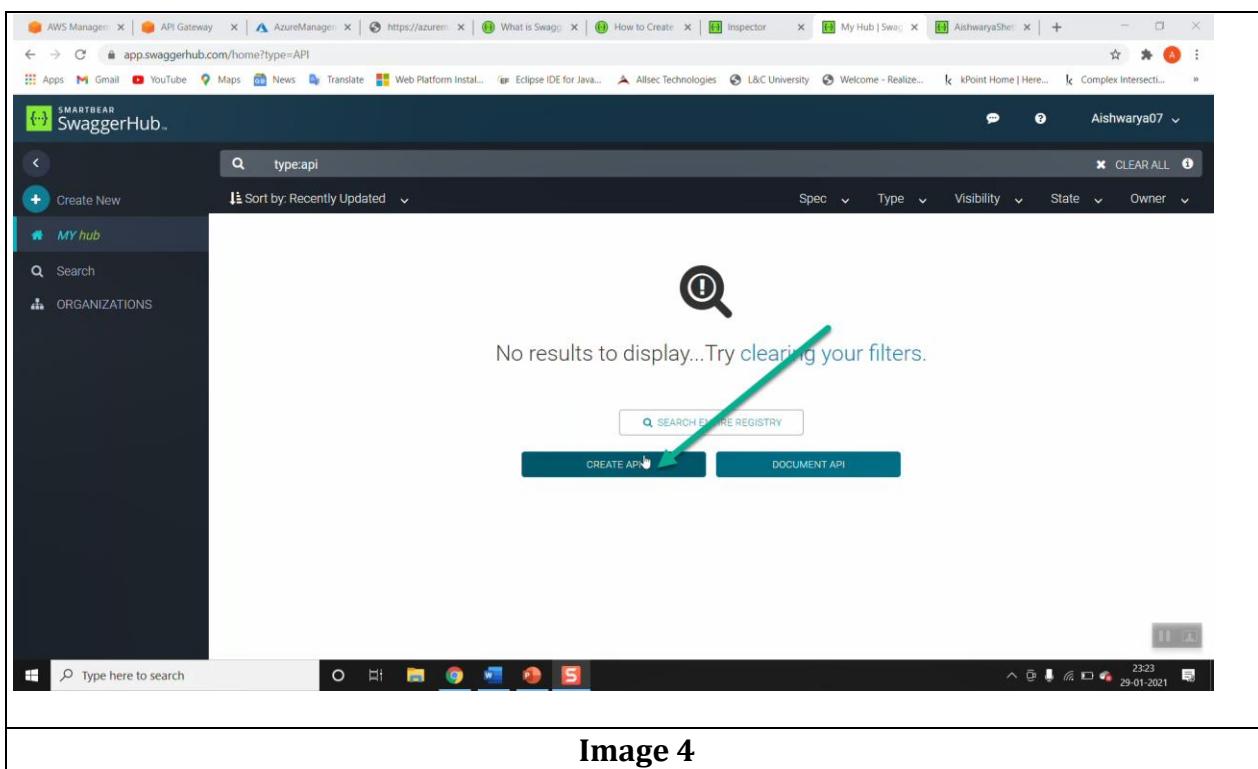


Image 4

Step 6- Then fill the Form which includes the version, name and other details click on Create API button. As depicted in image 5.

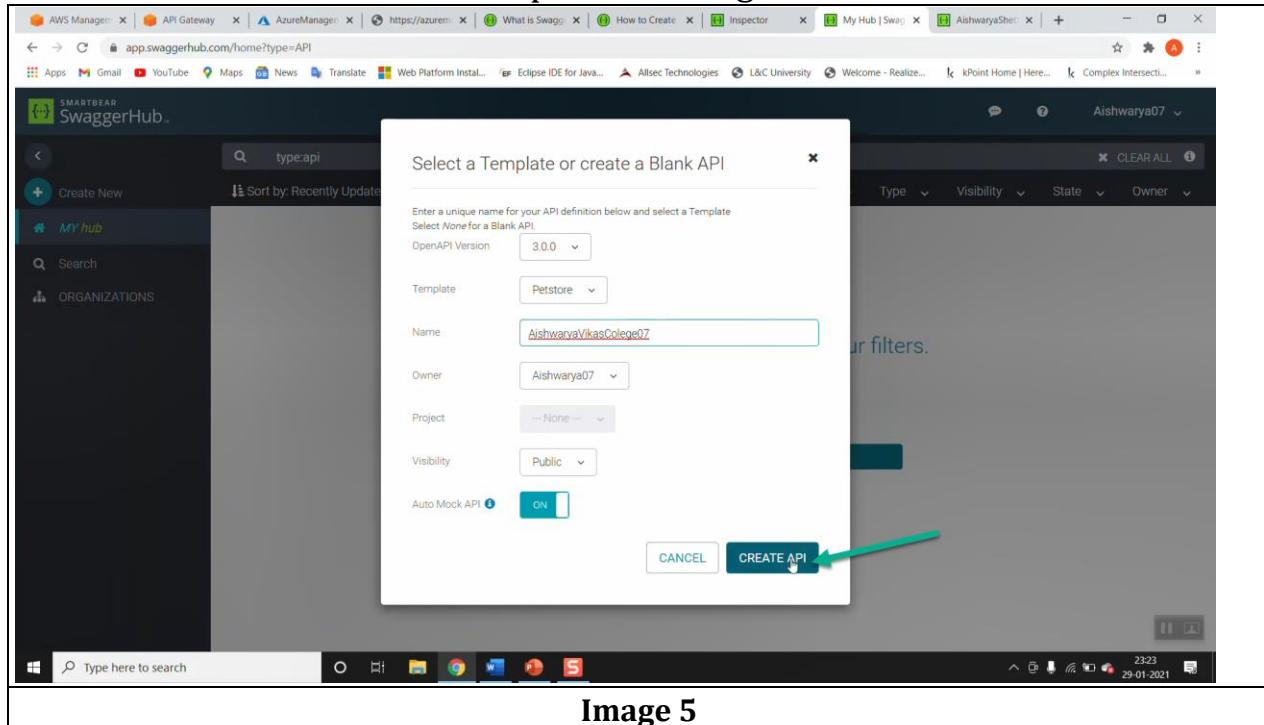


Image 5

Step 7- Copy the API and definition from swagger as shown in image 6

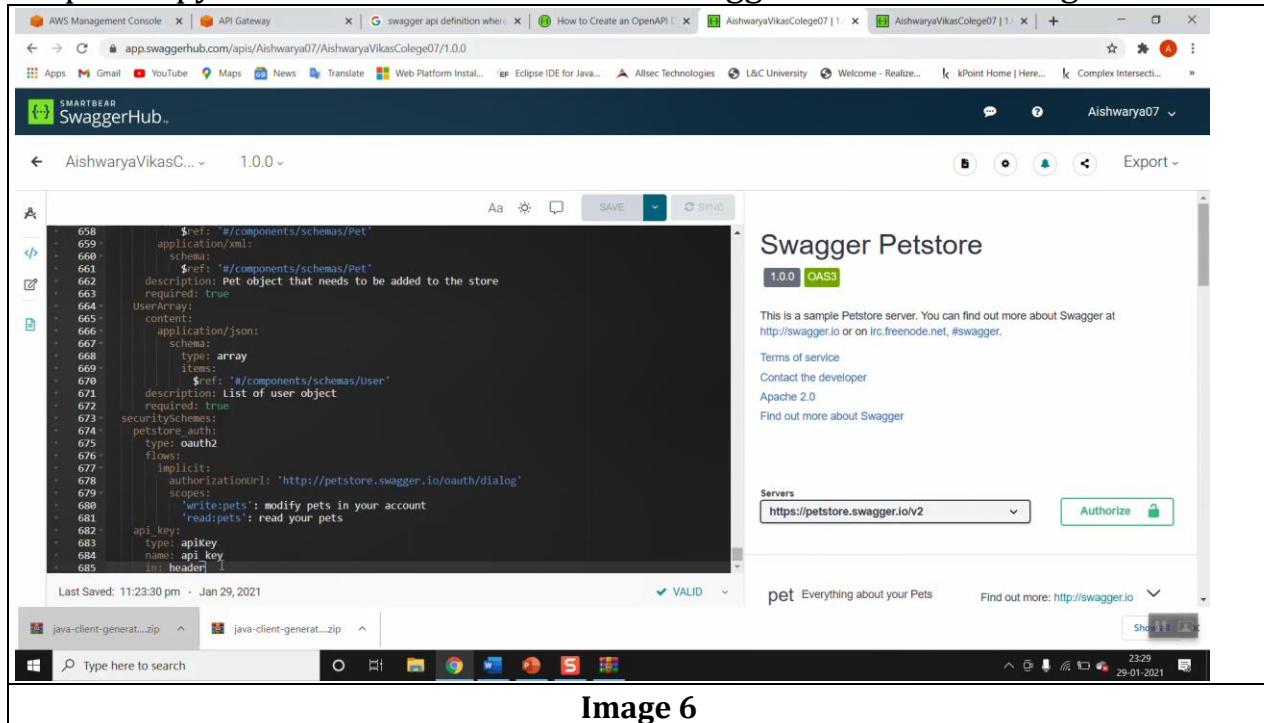
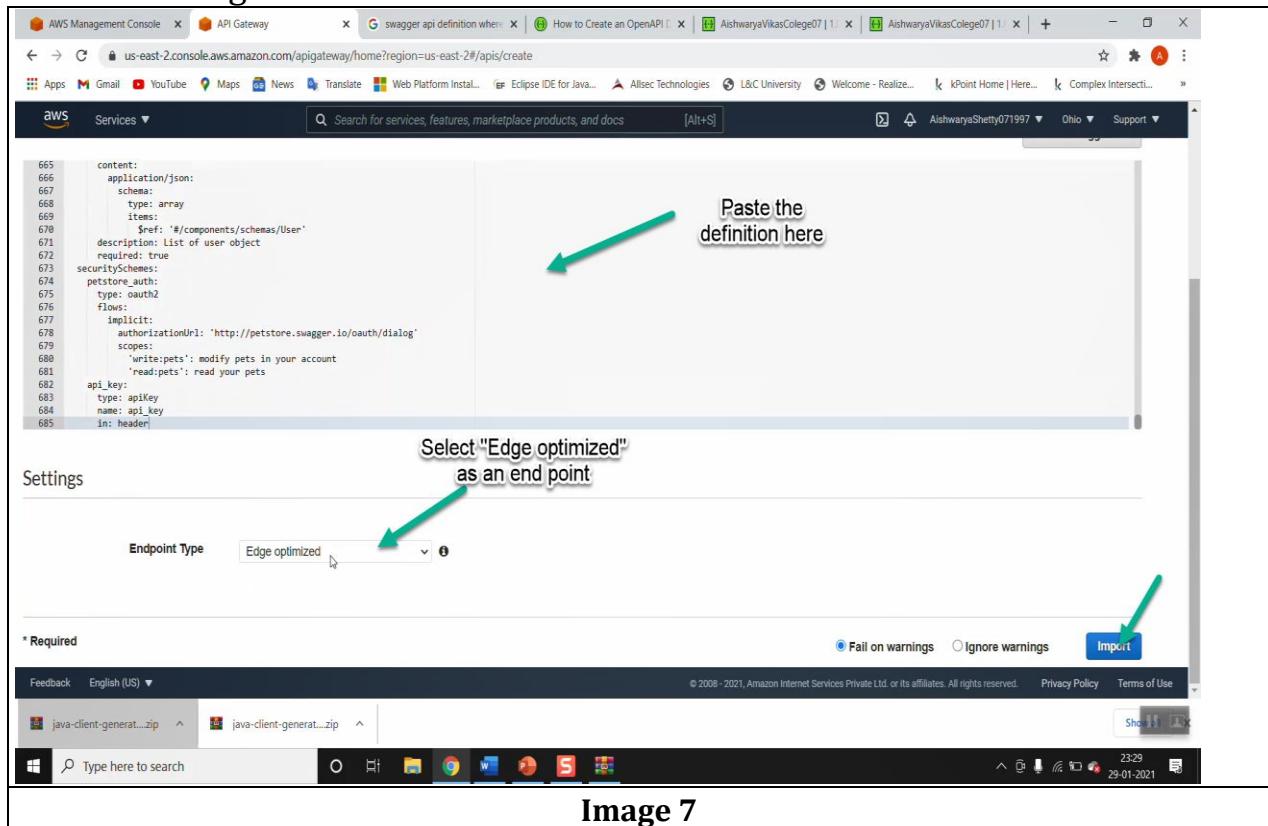


Image 6

Step 8-Paste the definition and in In the Settings section, select the endpoint type. Select “Edge Optimized” option and then click on import button as

shown in image 7



API definition from swagger is mentioned below-

openapi: 3.0.0

info:

description: |

This is a sample Petstore server. You can find out more about Swagger at http://swagger.io or on [irc.freenode.net, #swagger](http://swagger.io/irc/).

version: "1.0.0"

title: Swagger Petstore

termsOfService: 'http://swagger.io/terms/'

contact:

email: apiteam@swagger.io

license:

name: Apache 2.0

url: 'http://www.apache.org/licenses/LICENSE-2.0.html'

servers:

Added by API Auto Mocking Plugin

- description: SwaggerHub API Auto Mocking

```
url:  
https://virtserver.swaggerhub.com/Aishwarya07/AishwaryaVikasColege07/1  
.0.0  
- url: 'https://petstore.swagger.io/v2'  
tags:  
- name: pet  
description: Everything about your Pets  
externalDocs:  
description: Find out more  
url: 'http://swagger.io'  
- name: store  
description: Access to Petstore orders  
- name: user  
description: Operations about user  
externalDocs:  
description: Find out more about our store  
url: 'http://swagger.io'  
paths:  
/pet:  
post:  
tags:  
- pet  
summary: Add a new pet to the store  
operationId: addPet  
responses:  
'405':  
description: Invalid input  
security:  
- petstore_auth:  
- 'write:pets'  
- 'read:pets'  
requestBody:  
$ref: '#/components/requestBodies/Pet'  
put:  
tags:  
- pet  
summary: Update an existing pet  
operationId: updatePet  
responses:
```

```
'400':  
    description: Invalid ID supplied  
'404':  
    description: Pet not found  
'405':  
    description: Validation exception  
security:  
  - petstore_auth:  
    - 'write:pets'  
    - 'read:pets'  
requestBody:  
  $ref: '#/components/requestBodies/Pet'  
/pet/findByStatus:  
get:  
tags:  
  - pet  
summary: Finds Pets by status  
description: Multiple status values can be provided with comma separated strings  
operationId: findPetsByStatus  
parameters:  
  - name: status  
    in: query  
    description: Status values that need to be considered for filter  
    required: true  
    explode: true  
    schema:  
      type: array  
      items:  
        type: string  
        enum:  
          - available  
          - pending  
          - sold  
        default: available  
responses:  
'200':  
  description: successful operation  
  content:
```

```
application/json:
  schema:
    type: array
    items:
      $ref: '#/components/schemas/Pet'
application/xml:
  schema:
    type: array
    items:
      $ref: '#/components/schemas/Pet'
'400':
  description: Invalid status value
security:
  - petstore_auth:
    - 'write:pets'
    - 'read:pets'
/pet/findByTags:
get:
  tags:
    - pet
summary: Finds Pets by tags
description: >-
  Muliple tags can be provided with comma separated strings. Use\\ \\ tag1,
  tag2, tag3 for testing.
operationId: findPetsByTags
parameters:
  - name: tags
    in: query
    description: Tags to filter by
    required: true
    explode: true
    schema:
      type: array
      items:
        type: string
responses:
  '200':
    description: successful operation
    content:
```

```
application/json:
  schema:
    type: array
    items:
      $ref: '#/components/schemas/Pet'
application/xml:
  schema:
    type: array
    items:
      $ref: '#/components/schemas/Pet'
'400':
  description: Invalid tag value
security:
  - petstore_auth:
    - 'write:pets'
    - 'read:pets'
  deprecated: true
'/pet/{petId}':
  get:
    tags:
      - pet
    summary: Find pet by ID
    description: Returns a single pet
    operationId: getPetById
    parameters:
      - name: petId
        in: path
        description: ID of pet to return
        required: true
        schema:
          type: integer
          format: int64
    responses:
      '200':
        description: successful operation
        content:
          application/json:
            schema:
              $ref: '#/components/schemas/Pet'
```

```
application/xml:  
  schema:  
    $ref: '#/components/schemas/Pet'  
'400':  
  description: Invalid ID supplied  
'404':  
  description: Pet not found  
security:  
  - api_key: []  
post:  
tags:  
  - pet  
summary: Updates a pet in the store with form data  
operationId: updatePetWithForm  
parameters:  
  - name: petId  
    in: path  
    description: ID of pet that needs to be updated  
    required: true  
    schema:  
      type: integer  
      format: int64  
responses:  
'405':  
  description: Invalid input  
security:  
  - petstore_auth:  
    - 'write:pets'  
    - 'read:pets'  
requestBody:  
content:  
  application/x-www-form-urlencoded:  
    schema:  
      type: object  
      properties:  
        name:  
          description: Updated name of the pet  
          type: string  
        status:
```

```
    description: Updated status of the pet
    type: string
delete:
tags:
- pet
summary: Deletes a pet
operationId: deletePet
parameters:
- name: api_key
  in: header
  required: false
  schema:
    type: string
- name: petId
  in: path
  description: Pet id to delete
  required: true
  schema:
    type: integer
    format: int64
responses:
'400':
  description: Invalid ID supplied
'404':
  description: Pet not found
security:
- petstore_auth:
  - 'write:pets'
  - 'read:pets'
'/pet/{petId}/uploadImage':
post:
tags:
- pet
summary: uploads an image
operationId: uploadFile
parameters:
- name: petId
  in: path
  description: ID of pet to update
```

```
required: true
schema:
  type: integer
  format: int64
responses:
  '200':
    description: successful operation
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/ApiResponse'
security:
  - petstore_auth:
    - 'write:pets'
    - 'read:pets'
requestBody:
  content:
    application/octet-stream:
      schema:
        type: string
        format: binary
/store/inventory:
get:
  tags:
    - store
  summary: Returns pet inventories by status
  description: Returns a map of status codes to quantities
  operationId: getInventory
  responses:
    '200':
      description: successful operation
      content:
        application/json:
          schema:
            type: object
            additionalProperties:
              type: integer
              format: int32
  security:
```

```
- api_key: []
/store/order:
post:
tags:
- store
summary: Place an order for a pet
operationId: placeOrder
responses:
'200':
description: successful operation
content:
application/json:
schema:
$ref: '#/components/schemas/Order'
application/xml:
schema:
$ref: '#/components/schemas/Order'
'400':
description: Invalid Order
requestBody:
content:
application/json:
schema:
$ref: '#/components/schemas/Order'
description: order placed for purchasing the pet
required: true
'/store/order/{orderId}':
get:
tags:
- store
summary: Find purchase order by ID
description: >-
For valid response try integer IDs with value >= 1 and <= 10.\ \
Other values will generated exceptions
operationId: getOrderBy
parameters:
- name: orderId
in: path
description: ID of pet that needs to be fetched
```

```
required: true
schema:
  type: integer
  format: int64
  minimum: 1
  maximum: 10
responses:
  '200':
    description: successful operation
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/Order'
      application/xml:
        schema:
          $ref: '#/components/schemas/Order'
  '400':
    description: Invalid ID supplied
  '404':
    description: Order not found
delete:
tags:
  - store
summary: Delete purchase order by ID
description: >-
  For valid response try integer IDs with positive integer value.\\ \
  Negative or non-integer values will generate API errors
operationId: deleteOrder
parameters:
  - name: orderId
    in: path
    description: ID of the order that needs to be deleted
    required: true
    schema:
      type: integer
      format: int64
      minimum: 1
responses:
  '400':
```

```
    description: Invalid ID supplied
    '404':
        description: Order not found
/user:
post:
tags:
- user
summary: Create user
description: This can only be done by the logged in user.
operationId: createUser
responses:
default:
    description: successful operation
requestBody:
content:
application/json:
schema:
$ref: '#/components/schemas/User'
description: Created user object
required: true
/user/createWithArray:
post:
tags:
- user
summary: Creates list of users with given input array
operationId: createUsersWithArrayInput
responses:
default:
    description: successful operation
requestBody:
$ref: '#/components/requestBodies/UserArray'
/user/createWithList:
post:
tags:
- user
summary: Creates list of users with given input array
operationId: createUsersWithListInput
responses:
default:
```

```
    description: successful operation
    requestBody:
      $ref: '#/components/requestBodies/UserArray'
  /user/login:
    get:
      tags:
        - user
      summary: Logs user into the system
      operationId: loginUser
      parameters:
        - name: username
          in: query
          description: The user name for login
          required: true
          schema:
            type: string
        - name: password
          in: query
          description: The password for login in clear text
          required: true
          schema:
            type: string
      responses:
        '200':
          description: successful operation
          headers:
            X-Rate-Limit:
              description: calls per hour allowed by the user
              schema:
                type: integer
                format: int32
            X-Expires-After:
              description: date in UTC when token expires
              schema:
                type: string
                format: date-time
          content:
            application/json:
              schema:
```

```
    type: string
  application/xml:
    schema:
      type: string
  '400':
    description: Invalid username/password supplied
/user/logout:
get:
tags:
- user
summary: Logs out current logged in user session
operationId: logoutUser
responses:
default:
  description: successful operation
'/user/{username}':
get:
tags:
- user
summary: Get user by user name
operationId: getUserByName
parameters:
- name: username
  in: path
  description: The name that needs to be fetched. Use user1 for testing.
  required: true
  schema:
    type: string
responses:
'200':
  description: successful operation
  content:
    application/json:
      schema:
        $ref: '#/components/schemas/User'
    application/xml:
      schema:
        $ref: '#/components/schemas/User'
  '400':
```

```
    description: Invalid username supplied
    '404':
        description: User not found
  put:
    tags:
      - user
    summary: Updated user
    description: This can only be done by the logged in user.
    operationId: updateUser
    parameters:
      - name: username
        in: path
        description: name that need to be updated
        required: true
      schema:
        type: string
    responses:
      '400':
        description: Invalid user supplied
      '404':
        description: User not found
  requestBody:
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/User'
    description: Updated user object
    required: true
  delete:
    tags:
      - user
    summary: Delete user
    description: This can only be done by the logged in user.
    operationId: deleteUser
    parameters:
      - name: username
        in: path
        description: The name that needs to be deleted
        required: true
```

```
schema:
  type: string
responses:
  '400':
    description: Invalid username supplied
  '404':
    description: User not found
externalDocs:
  description: Find out more about Swagger
  url: 'http://swagger.io'
components:
  schemas:
    Order:
      type: object
      properties:
        id:
          type: integer
          format: int64
        petId:
          type: integer
          format: int64
        quantity:
          type: integer
          format: int32
        shipDate:
          type: string
          format: date-time
        status:
          type: string
          description: Order Status
          enum:
            - placed
            - approved
            - delivered
        complete:
          type: boolean
          default: false
      xml:
        name: Order
```

```
Category:  
  type: object  
  properties:  
    id:  
      type: integer  
      format: int64  
    name:  
      type: string  
    xml:  
      name: Category  
User:  
  type: object  
  properties:  
    id:  
      type: integer  
      format: int64  
    username:  
      type: string  
    firstName:  
      type: string  
    lastName:  
      type: string  
    email:  
      type: string  
    password:  
      type: string  
    phone:  
      type: string  
    userStatus:  
      type: integer  
      format: int32  
      description: User Status  
    xml:  
      name: User  
Tag:  
  type: object  
  properties:  
    id:  
      type: integer
```

```
format: int64
name:
  type: string
xml:
  name: Tag
Pet:
  type: object
  required:
    - name
    - photoUrls
  properties:
    id:
      type: integer
      format: int64
    category:
      $ref: '#/components/schemas/Category'
    name:
      type: string
      example: doggie
    photoUrls:
      type: array
      xml:
        name: photoUrl
        wrapped: true
      items:
        type: string
    tags:
      type: array
      xml:
        name: tag
        wrapped: true
      items:
        $ref: '#/components/schemas/Tag'
    status:
      type: string
      description: pet status in the store
      enum:
        - available
        - pending
```

```
- sold
xml:
  name: Pet
ApiResponse:
  type: object
  properties:
    code:
      type: integer
      format: int32
    type:
      type: string
    message:
      type: string
requestBodies:
  Pet:
    content:
      application/json:
        schema:
          $ref: '#/components/schemas/Pet'
      application/xml:
        schema:
          $ref: '#/components/schemas/Pet'
    description: Pet object that needs to be added to the store
    required: true
  UserArray:
    content:
      application/json:
        schema:
          type: array
          items:
            $ref: '#/components/schemas/User'
    description: List of user object
    required: true
  securitySchemes:
    petstore_auth:
      type: oauth2
      flows:
        implicit:
          authorizationUrl: 'http://petstore.swagger.io/oauth/dialog'
```

scopes:

'write:pets': modify pets in your account

'read:pets': read your pets

api_key:

type: apiKey

name: api_key

in: header

Output-

If the import is successful, we can view the "**API Structure in AWS API Gateway**" where we can see methods of the API. As shown in image 8

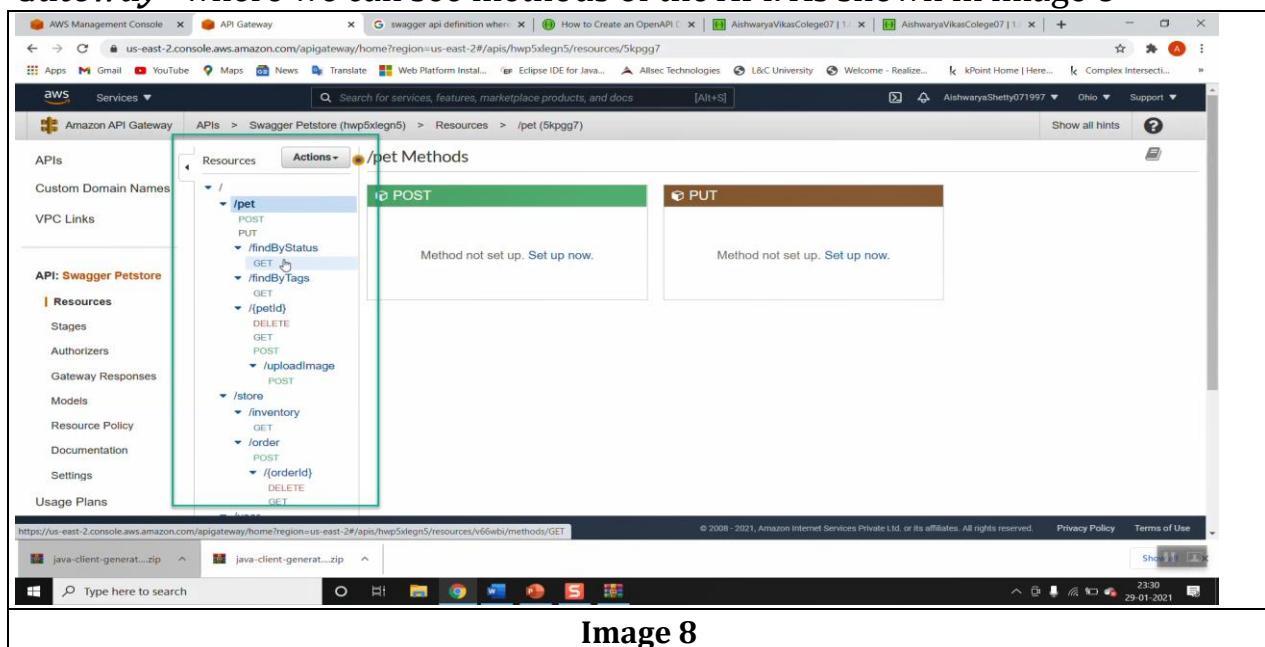


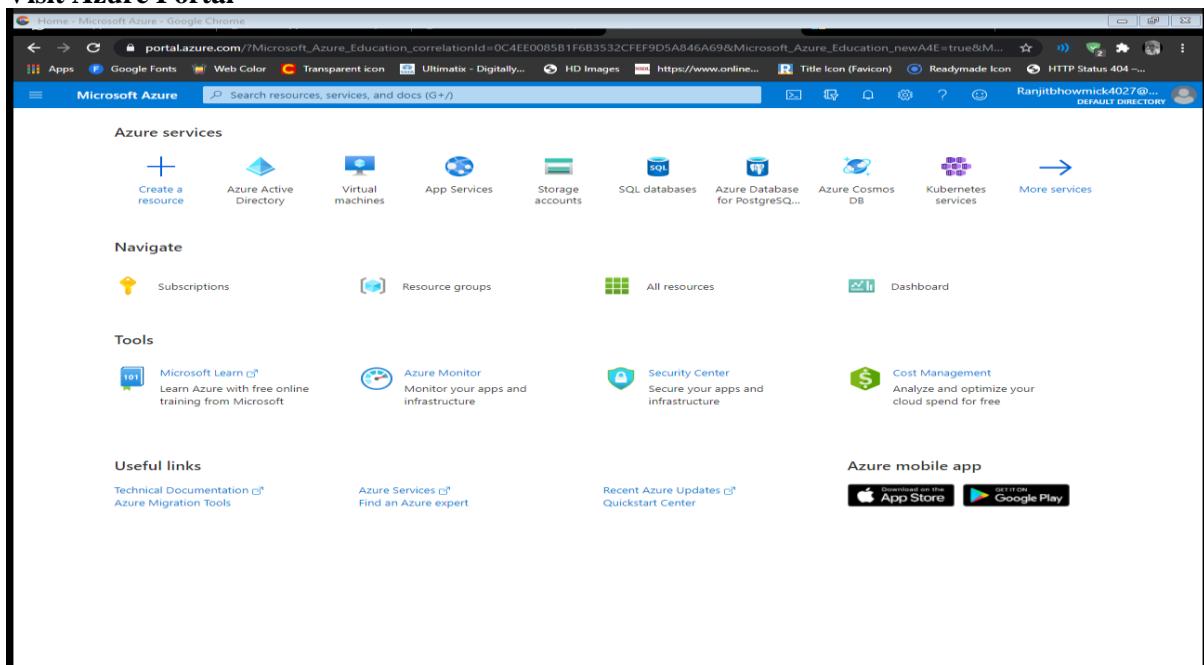
Image 8

PRACTICAL 9

Securing APIs with Azure Active Directory

1. Register an application in Azure AD to represent the API

Visit Azure Portal -



Search App Registration-

The screenshot shows the Microsoft Azure portal interface. The left sidebar includes sections for 'Create a resource', 'Navigate', 'Tools', and 'Useful links'. The main content area has a search bar at the top with the query 'App registration'. Below the search bar, there's a 'Services' section listing 'App registrations', 'App Services', 'App Service Certificates', 'App Service Domains', 'App Service Environments', 'App Service plans', 'AppDynamics', 'Appliances', 'App Configuration', and 'App proxy'. A 'Marketplace' section shows 'No results were found.' A 'Documentation' section contains links to 'Quickstart: Register an app in the Microsoft identity ...', 'Create an Azure AD app & service principal in the portal ...', 'Tutorial: Register a single-page application - Azure AD ...', and 'Custom role permissions for app registration - Azure AD ...'. A 'Resource Groups' section also shows 'No results were found.' At the bottom of the page, there are links for 'Try searching in Activity Log' and 'Try searching in Azure Active Directory', along with a note about 'Searching all subscriptions. Change'. On the right side, there's a 'More services' button with an arrow icon. At the bottom right, there are links for the 'Azure mobile app' available on the App Store and Google Play.

Click on New Registration-

The screenshot shows the Microsoft Azure App registrations page. At the top, there are several informational banners: one about the new search preview, another about the deprecation of ADAL and Azure AD Graph starting June 30th, 2020, and a third about the transition to MSAL and Microsoft Graph. Below these, there are tabs for 'All applications', 'Owned applications' (which is selected), and 'Applications from personal account'. A search bar is present with the placeholder 'Start typing a name or Application ID to filter these results'. A message indicates that the current user account isn't listed as an owner of any applications. Two buttons are available: 'View all applications in the directory' and 'View all applications from personal account'.

Provide proper name and click on Register –

The screenshot shows the Microsoft Azure Register an application page. The first step is to enter the application's name, which is 'KJ-WeatherForecast'. The next section, 'Supported account types', includes a question 'Who can use this application or access this API?' and four options: 'Accounts in this organizational directory only (Default Directory only - Single tenant)' (selected with a radio button), 'Accounts in any organizational directory (Any Azure AD directory - Multitenant)', 'Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)', and 'Personal Microsoft accounts only'. There is also a 'Help me choose...' link. The 'Redirect URI (optional)' section contains a dropdown menu set to 'Web' and a text input field containing 'e.g. https://example.com/auth'. At the bottom, there is a note about adding apps from outside the organization and a link to 'Enterprise applications'. A checkbox for accepting platform policies is present, followed by a 'Register' button.

It will redirects to Overview Page –

The screenshot shows the Microsoft Azure portal interface. The URL in the address bar is https://portal.azure.com/#Microsoft_Azure_Education_correlationId=0C4EE0085B1F6B3532CFEF9D5A846A69&Microsoft_Azure_Education_newA4E=true&M.... The title bar says "KJ-WeatherForecast - Microsoft Azure - Google Chrome". The main content area is titled "KJ-WeatherForecast" and shows the "Overview" tab selected. The left sidebar has sections like "Manage", "Call APIs", and "Documentation". The "Essentials" section displays details such as Display name (KJ-WeatherForecast), Application (client) ID (36410ea7-b6f6-4104-9d20-e97572defbd9), Directory (tenant) ID (5768db3a-7236-4687-a45a-8b54c10057fa), Object ID (63131091-8c26-4022-8dae-287e29bc6fe1), Supported account types (My organization only), and Redirect URLs (Add a Redirect URI). There are also two informational messages about the new App registrations experience and the end of support for ADAL and Graph.

Select to Expose an API –

The screenshot shows the Microsoft Azure portal interface. The URL in the address bar is https://portal.azure.com/#Microsoft_Azure_Education_correlationId=0C4EE0085B1F6B3532CFEF9D5A846A69&Microsoft_Azure_Education_newA4E=true&M.... The page title is "KJ-WeatherForecast | Expose an API".

The left sidebar menu includes:

- Overview
- Quickstart
- Integration assistant
- Manage
 - Branding
 - Authentication
 - Certificates & secrets
 - Token configuration
 - API permissions
 - Expose an API**
 - App roles | Preview
 - Owners
 - Roles and administrators | Preview
 - Manifest
- Support + Troubleshooting
 - Troubleshooting
 - New support request

The main content area is titled "Scopes defined by this API". It contains a note: "Define custom scopes to restrict access to data and functionality protected by the API. An application that requires access to parts of this API can request that a user or admin consent to one or more of these." Below this is a note: "Adding a scope here creates only delegated permissions. If you are looking to create application-only scopes, use 'App roles' and define app roles assignable to application type. [Go to App roles.](#)". There is a button "+ Add a scope". A table header row is shown with columns: Scopes, Who can consent, Admin consent displ..., User consent displ..., and State. The table body states "No scopes have been defined".

Below this is the "Authorized client applications" section. It contains a note: "Authorizing a client application indicates that this API trusts the application and users should not be asked to consent when the client calls this API." There is a button "+ Add a client application". A table header row is shown with columns: Client Id and Scopes. The table body states "No client applications have been authorized".

Set Default App URL-

The screenshot shows the Microsoft Azure portal interface. The left sidebar has a tree view with 'Expose an API' selected under 'API permissions'. The main content area is titled 'KJ-WeatherForecast | Expose an API'. It shows an 'Application ID URI' of 'api://36410ea7-b6f6-4104-9d20-e97573defbd9'. A table titled 'Scopes defined by this API' is present, with a single row indicating 'No scopes have been defined'. Below this, the 'Authorized client applications' section shows 'No client applications have been authorized'.

Set value in opened scope page –

This screenshot shows the 'Edit a scope' dialog box overlaid on the main Azure portal page. The dialog has a title 'Edit a scope' and contains fields for 'Scope name' (set to 'WeatherForecastRead'), 'Who can consent' (set to 'Admins and users'), 'Admin consent display name' (set to 'WeatherForecastRead'), 'Admin consent description' (set to 'WeatherForecast'), 'User consent display name' (set to 'WeatherForecastRead'), 'User consent description' (set to 'WeatherForecastRead'), and a 'State' dropdown set to 'Enabled'. The background page remains largely the same as the first screenshot, with the 'Expose an API' section visible.

Register an application in Azure AD to represent a client Application – Navigate to Azure Home Page –

The screenshot shows the Microsoft Azure Home Page. At the top, there's a search bar and a user profile. Below the header, there's a section titled "Azure services" with icons for Create a resource, App registrations, Azure Active Directory, Virtual machines, App Services, Storage accounts, SQL databases, Azure Database for PostgreSQL, Azure Cosmos DB, and More services. Under "Navigate", there are links for Subscriptions, Resource groups, All resources, and Dashboard. In the "Tools" section, there are links for Microsoft Learn, Azure Monitor, Security Center, and Cost Management. The "Useful links" section includes Technical Documentation, Azure Services, Recent Azure Updates, and Quickstart Center. The "Azure mobile app" section shows download links for the App Store and Google Play.

Again search App Registration –

The screenshot shows the Microsoft Azure App registrations page. At the top, there's a search bar and a user profile. Below the header, there's a section titled "App registrations" with links for New registration, Endpoints, Troubleshooting, Download, Preview features, and Got feedback?. There are two informational messages: one about the preview feature and another about the end of support for ADAL and Azure AD Graph. Below these, there are tabs for All applications, Owned applications (which is selected), and Applications from personal account. A search bar allows filtering by name or Application ID. A table lists the registered application "KJ-WeatherForecast".

| Display name | Application (client) ID | Created on | Certificates & secrets |
|--------------------|--------------------------------------|------------|------------------------|
| KJ-WeatherForecast | 36410ea7-b6f6-4104-9d20-e97573defbd9 | 1/30/2021 | - |

Click on New Registration

The screenshot shows the Microsoft Azure portal's 'Register an application' interface. At the top, there's a navigation bar with links like 'Home', 'App registrations', and 'Search resources, services, and docs (G+)'. Below the navigation is a form titled 'Register an application'. The first field, 'Name', is required and has a placeholder 'The user-facing display name for this application (this can be changed later)'. A value 'KJ-WeatherForecastClient' is entered. The next section, 'Supported account types', contains four options: 'Accounts in this organizational directory only (Default Directory only - Single tenant)' (radio button), 'Accounts in any organizational directory (Any Azure AD directory - Multitenant)' (radio button, selected), 'Accounts in any organizational directory (Any Azure AD directory - Multitenant) and personal Microsoft accounts (e.g. Skype, Xbox)' (radio button), and 'Personal Microsoft accounts only' (radio button). Below this is a link 'Help me choose...'. The 'Redirect URI (optional)' section includes a dropdown set to 'Web' and a text input field containing 'e.g. https://example.com/auth'. At the bottom of the form, there's a note about integrating apps from outside the organization and a link to 'Enterprise applications'. A 'By proceeding, you agree to the Microsoft Platform Policies' checkbox is present. Finally, a large blue 'Register' button is at the bottom.

Now click on Register-

This screenshot shows the same 'Register an application' page as the previous one, but it has been submitted. The 'Register' button is now greyed out, indicating the process is complete. The rest of the page content remains the same, including the filled-in 'Name' field, selected 'Supported account types', and the optional 'Redirect URI' field.

You will redirect to this page –

The screenshot shows the Microsoft Azure portal with the URL https://portal.azure.com/#Microsoft_Azure_KJ-WeatherForecastClient. The page title is "KJ-WeatherForecastClient". The left sidebar includes links for Overview, Quickstart, Integration assistant, Manage (Branding, Authentication, Certificates & secrets, Token configuration, API permissions, Expose an API, App roles | Preview, Owners, Roles and administrators | Preview, Manifest), Support + Troubleshooting (Troubleshooting, New support request), and Call APIs (with icons for various services like Cloud, X, S, N, etc.). The main content area displays the application's details under the "Essentials" section, including Display name (KJ-WeatherForecastClient), Application (client) ID (52b3bf4f-c1e4-4aba-b243-48650be2d6e5), Directory (tenant) ID (5768db3a-7236-4887-a45a-8b54c10057fa), Object ID (a9747b14-1e17-40e6-9ab2-05b66d0d3229), Supported account types (Multiple organizations), Redirect URIs (Add a Redirect URI), Application ID URI (Add an Application ID URI), and Managed application in local directory (KJ-WeatherForecastClient). There are also three informational banners: one about feedback for the Microsoft identity platform, one about changes to app registrations, and one about the end of user consent grants for unverified publishers.

Select Certificate and Secrets –

The screenshot shows the Microsoft Azure portal with the URL https://portal.azure.com/#Microsoft_Azure_KJ-WeatherForecastClient_Certificates_secrets. The page title is "KJ-WeatherForecastClient | Certificates & secrets". The left sidebar includes links for Overview, Quickstart, Integration assistant, Manage (Authentication, Certificates & secrets, Token configuration, API permissions, Expose an API, App roles | Preview, Owners, Roles and administrators | Preview, Manifest), Support + Troubleshooting (Troubleshooting, New support request), and Got feedback? (link to feedback form). The main content area shows the "Certificates" section, which states that certificates can be used as secrets to prove the application's identity. It includes a "Upload certificate" button and a table for managing certificates. The table has columns for Thumbprint, Start date, Expires, and ID. A note says "No certificates have been added for this application." Below this is the "Client secrets" section, which states that a secret string proves the application's identity. It includes a "+ New client secret" button and a table for managing client secrets. The table has columns for Description, Expires, Value, and ID. A note says "No client secrets have been created for this application."

And select client Secret – Add Description and Expires details-

The screenshot shows the Microsoft Azure portal interface. The left sidebar navigation menu is visible, showing options like Overview, Quickstart, Integration assistant, Manage (Branding, Authentication, Certificates & secrets, Token configuration, API permissions, Expose an API, App roles | Preview, Owners, Roles and administrators | Preview, Manifest), Support + Troubleshooting (Troubleshooting, New support request), and a section for Troubleshooting and New support request.

The main content area displays a modal dialog titled "Add a client secret". The "Description" field contains "WeatherForecastClientSecret". The "Expires" section has three radio button options: "In 1 year" (selected), "In 2 years", and "Never". Below the dialog are sections for "Client secrets" and "New client secret". The "Client secrets" section includes a note: "A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password." The "New client secret" section has columns for Description, Expires, Value, and ID, with a note: "No client secrets have been created for this application."

KJ-WeatherForecastClient | Certificates & secrets

Overview Quickstart Integration assistant

Manage

Branding Authentication Certificates & secrets Token configuration API permissions Expose an API App roles | Preview Owners Roles and administrators | Preview Manifest

Support + Troubleshooting

Troubleshooting New support request

Copy the new client secret value. You won't be able to retrieve it after you perform another operation or leave this blade.

Credentials enable confidential applications to identify themselves to the authentication service when receiving tokens at a web addressable location (using an HTTPS scheme). For a higher level of assurance, we recommend using a certificate (instead of a client secret) as a credential.

Certificates

Certificates can be used as secrets to prove the application's identity when requesting a token. Also can be referred to as public keys.

Upload certificate

| Thumbprint | Start date | Expires | ID |
|---|------------|---------|----|
| No certificates have been added for this application. | | | |

Client secrets

A secret string that the application uses to prove its identity when requesting a token. Also can be referred to as application password.

New client secret

| Description | Expires | Value | ID |
|-----------------------------|-----------|--|--------------------------------------|
| WeatherForecastClientSecret | 1/30/2022 | H~UdF5mHlg9~1Or9nBkJ5AGWx9ud_Yj7_-5ce7ea6d-c281-4079-9785-9c1c259b5b26 | 5ce7ea6d-c281-4079-9785-9c1c259b5b26 |

WeatherForecastClientSecret

1/30/2022

H~UdF5mHlg9~1Or9nBkJ5AGWx9ud_Yj7_-
5ce7ea6d-c281-4079-9785-9c1c259b5b26

Grant Permission in Azure AD – Navigate to Azure Portal –

The screenshot shows the Microsoft Azure Home page. At the top, there's a search bar with placeholder text "Search resources, services, and docs (G+/" and a magnifying glass icon. Below the search bar, the "Azure services" section is visible, featuring a "Create a resource" button and a "Navigate" sidebar with "App registrations" and "Azure Active Directory". The "Recent services" section lists "App reg" and "Azure active". To the right, there are links for "Databases", "Azure Database for PostgreSQL...", "Azure Cosmos DB", and "More services". The "Tools" section includes links for "Microsoft Learn", "Azure Monitor", "Security Center", and "Cost Management". The "Useful links" section has links for "Technical Documentation", "Azure Services", "Recent Azure Updates", and "Azure mobile app" download links for the App Store and Google Play. The "App registrations" section shows a preview message about the new search feature and a note about the end of support for ADAL and Azure AD Graph. It also displays tabs for "All applications", "Owned applications" (which is selected), and "Applications from personal account". A search bar at the bottom allows filtering by "Display name" and "Application (client) ID".

Choose your client app and grant permission –

App registrations - Microsoft Azure - Google Chrome

portal.azure.com/Microsoft_Azure_Education_correlationId=0C4EE0085B1F6B3532CFEF9D5A846A69&Microsoft_Azure_Education_newA4E=true&M...

Microsoft Azure Search resources, services, and docs (G+)

Home > App registrations

New registration Endpoints Troubleshooting Download Preview features Got feedback?

Try out the new App registrations search preview! Click to enable the preview. →

Starting June 30th, 2020 we will no longer add any new features to Azure Active Directory Authentication Library (ADAL) and Azure AD Graph. We will continue to provide technical support and security updates but we will no longer provide feature updates. Applications will need to be upgraded to Microsoft Authentication Library (MSAL) and Microsoft Graph. Learn more

All applications Owned applications Applications from personal account

Start typing a name or Application ID to filter these results

| Display name | Application (client) ID | Created on | Certificates & secrets |
|--------------------------|--------------------------------------|------------|------------------------|
| KJ-WeatherForecast | 36410ea7-b6f6-4104-9d20-e97573defbd9 | 1/30/2021 | - |
| KJ-WeatherForecastClient | 52b3bf4f-c1e4-4aba-b243-48650be2d6e5 | 1/30/2021 | Current |

Select View All permission –

KJ-WeatherForecastClient - Microsoft Azure - Google Chrome

portal.azure.com/Microsoft_Azure_Education_correlationId=0C4EE0085B1F6B3532CFEF9D5A846A69&Microsoft_Azure_Education_newA4E=true&M...

Microsoft Azure Search resources, services, and docs (G+)

Home > App registrations >

KJ-WeatherForecastClient

Search (Ctrl+ /) Delete Endpoints Preview features

Overview Quickstart Integration assistant

Got a second? We would love your feedback on Microsoft identity platform (previously Azure AD for developer). →

Call APIs

Build more powerful apps with rich user and business data from Microsoft services and your own company's data sources.

View API permissions

Documentation

Microsoft identity platform
Authentication scenarios
Authentication libraries
Code samples
Microsoft Graph
Glossary
Help and Support

Sign in users in 5 minutes

Use our SDKs to sign in users and call APIs in a few steps

View all quickstart guides

Click on Add permission –

The screenshot shows the Microsoft Azure portal interface. The left sidebar navigation bar includes links for Overview, Quickstart, Integration assistant, Manage (with sub-links for Branding, Authentication, Certificates & secrets, Token configuration, API permissions, Expose an API, App roles | Preview, Owners, Roles and administrators | Preview, and Manifest), Support + Troubleshooting (with sub-links for Troubleshooting and New support request), and a Search bar.

The main content area displays the "KJ-WeatherForecastClient | API permissions" page. A warning message at the top states: "Starting November 9th, 2020 end users will no longer be able to grant consent to newly registered multitenant apps without verified publishers. Add MPN ID to verify publisher".

The "Configured permissions" section shows a table with one row:

| API / Permissions name | Type | Description | Admin consent req... | Status |
|------------------------|-----------|-------------|-------------------------------|--------|
| Microsoft Graph (1) | Delegated | User.Read | Sign in and read user profile | ... |

A note below the table says: "To view and manage permissions and user consent, try Enterprise applications."

Select on My Api and select your App-

The screenshot shows the Microsoft Azure portal interface, similar to the previous one but with a different URL in the address bar: "Request API permissions - Microsoft Azure - Google Chrome".

The left sidebar navigation bar is identical to the first screenshot.

The main content area displays the "Request API permissions" page for the "KJ-WeatherForecastClient | API permissions" section. It features a "Select an API" header with tabs for Microsoft APIs, APIs my organization uses, and My APIs (which is selected). A warning message "Starting November 9th, 2020" is displayed above the table.

The "Applications that expose permissions are shown below" section contains a table:

| Name | Application (client) ID |
|--------------------|--------------------------------------|
| KJ-WeatherForecast | 36410ea7-b6f6-4104-9d20-e97573defbd9 |

Add Required Permission under Delegated Permission –

The screenshot shows the 'Request API permissions' dialog in the Microsoft Azure portal. On the left, the navigation menu for the 'KJ-WeatherForecastClient' app registration is visible, showing sections like Overview, Quickstart, Integration assistant, Manage, Support + Troubleshooting, and more. The 'API permissions' section is currently selected. In the main pane, the 'Delegated permissions' section is active, displaying the 'WeatherForecastRead' permission. Below it, the 'Application permissions' section is shown, indicating that the application runs as a background service or daemon without a signed-in user. At the bottom, there are 'Add permissions' and 'Discard' buttons.

Page will look like this –

The screenshot shows the 'KJ-WeatherForecastClient | API permissions' page in the Microsoft Azure portal. The left sidebar shows the app registration details. The main area displays a message about successfully granted admin consent for the requested permissions. A warning message about end users no longer being able to grant consent to newly registered multitenant apps without verified publishers is present. The 'Configured permissions' section contains a table with the following data:

| API / Permissions name | Type | Description | Admin consent req... | Status |
|-------------------------|-----------|-------------------------------|----------------------|-----------------------------|
| WeatherForecastRead (1) | Delegated | WeatherForecastRead | - | Granted for Default Dire... |
| User.Read (1) | Delegated | Sign in and read user profile | - | Granted for Default Dire... |

Now enable OAuth 2.0 user Authorization – Navigate to Azure Homepage –

The screenshot shows the Microsoft Azure homepage. At the top, there's a search bar and a navigation bar with links like 'Home', 'Microsoft Azure', and 'Google Chrome'. Below the search bar, there's a banner with various service icons: Create a resource, App registrations, Azure Active Directory, Virtual machines, App Services, Storage accounts, SQL databases, Azure Database for PostgreSQL, Azure Cosmos DB, and More services. Under 'Navigate', there are links for Subscriptions, Resource groups, All resources, and Dashboard. In the 'Tools' section, there are links for Microsoft Learn, Azure Monitor, Security Center, and Cost Management. The 'Useful links' section includes Technical Documentation, Azure Services, Recent Azure Updates, and Quickstart Center. At the bottom right, there are links for the Azure mobile app on the App Store and Google Play.

Search API Management Instance –

API Management services - Microsoft Azure - Google Chrome

portal.azure.com/?Microsoft_Azure_Education_correlationId=0C4EE0085B1F6B3532CFEF9D5A846A69&Microsoft_Azure_Education_newA4E=true&M...

Apps Google Fonts Web Color Transparent icon Ultimax - Digitally... HD Images https://www.onlin... Title Icon (Favicon) Readymade Icon HTTP Status 404 ...

Microsoft Azure Search resources, services, and docs (G+) Ranjithbhowmick4027@... DEFAULT DIRECTORY

Home > API Management services

Default Directory

Add Manage view Refresh Export to CSV Open query Assign tags Feedback

Filter for any field... Subscription == all Resource group == all Location == all Add filter

No grouping List view

Showing 1 to 1 of 1 records.

| Name | Status | Tier | Type | Location | Resource group | Subscription |
|--------------------------|------------|-----------|------------------------|----------|--------------------------|--------------------|
| WeatherForcastManagement | Activating | Developer | API Management service | East US | WeatherForcastManagem... | Azure for Students |

< Previous Page 1 of 1 Next >

The screenshot shows the Azure API Management services list page. At the top, there's a navigation bar with links for Apps, Google Fonts, Web Color, Transparent icon, Ultimax - Digitally..., HD Images, Title Icon (Favicon), Readymade Icon, and HTTP Status 404 Below that is the Microsoft Azure logo and a search bar. The main title is "API Management services". Underneath, it says "Default Directory". There are buttons for "Add", "Manage view", "Refresh", "Export to CSV", "Open query", "Assign tags", and "Feedback". A search bar with placeholder "Filter for any field..." and dropdown filters for "Subscription == all", "Resource group == all", and "Location == all" are present. To the right are "No grouping" and "List view" buttons. The main area displays a single record in a table:

| Name | Status | Tier | Type | Location | Resource group | Subscription |
|--------------------------|------------|-----------|------------------------|----------|--------------------------|--------------------|
| WeatherForcastManagement | Activating | Developer | API Management service | East US | WeatherForcastManagem... | Azure for Students |

At the bottom, there are navigation links for "< Previous", "Page 1 of 1", and "Next >".

Select OAuth 2.0

WeatherForecastManagement - Microsoft Azure - Google Chrome
portal.azure.com/?Microsoft_Azure_Education_correlationId=0C4EE0085B1F6B3532CFEF9D5A846A69&Microsoft_Azure_Education_newA4E=true&M...
Apps Google Fonts Web Color Transparent icon Ultimatrix - Digitally... HD Images https://www.onlin... R Title Icon (Favicon) Readymade Icon HTTP Status 404 ...
Microsoft Azure Search resources, services, and docs (G+) Ranjithbhowmick4027@... DEFAULT DIRECTORY

Home > API Management services > WeatherForecastManagement

API Management service

WeatherForecastManagement | OAuth 2.0 + OpenID Connect

OAuth 2.0 OpenID Connect

OAuth 2.0

+ Add Columns Refresh

Search to filter items...

| Display name | Description |
|--------------|-------------|
| No results | |

Filter for any field...

Name ↑

WeatherForecastManagement

Developer portal

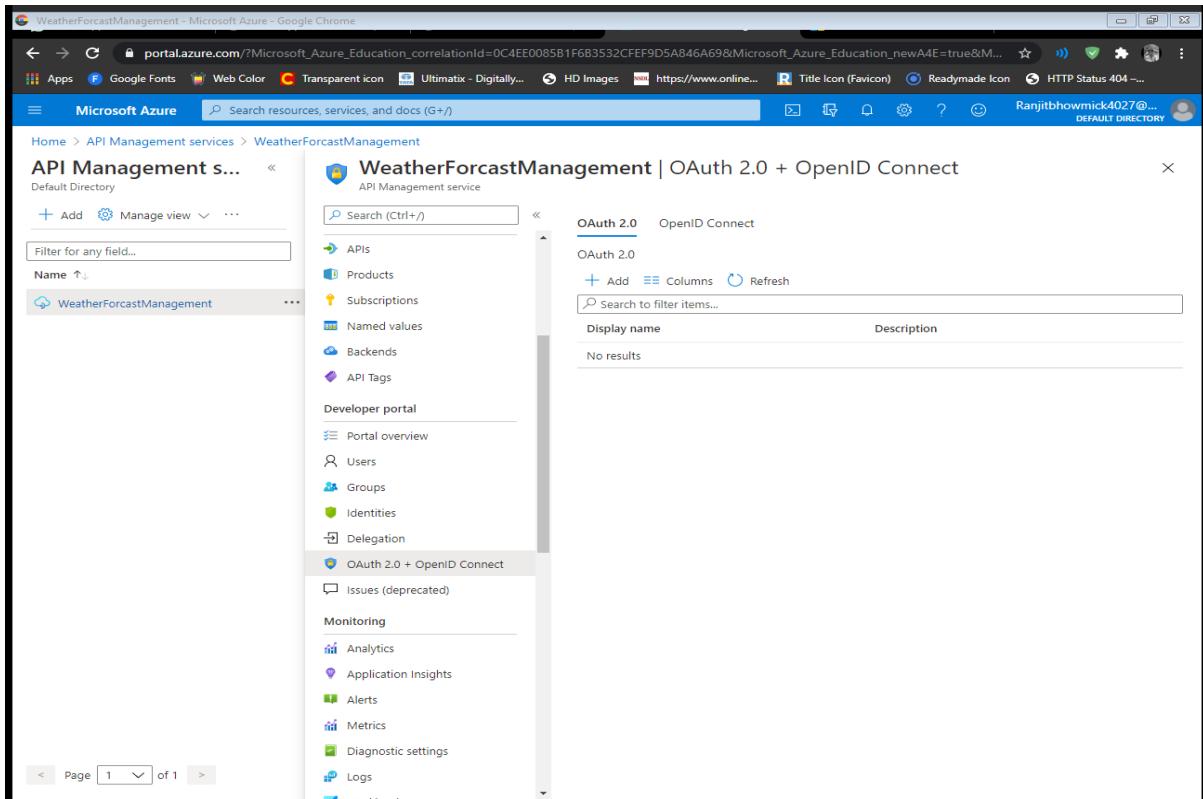
- Portal overview
- Users
- Groups
- Identities
- Delegation
- OAuth 2.0 + OpenID Connect**
- Issues (deprecated)

Monitoring

- Analytics
- Application Insights
- Alerts
- Metrics
- Diagnostic settings
- Logs

Add Manage view ...

Page 1 of 1



Add OAuth 2.0 –

Add OAuth2 service - Microsoft Azure - Google Chrome
portal.azure.com/?Microsoft_Azure_Education_correlationId=0C4EE0085B1F6B3532CFEF9D5A846A69&Microsoft_Azure_Education_newA4E=true&M...
Apps Google Fonts Web Color Transparent icon Ultimatrix - Digitally... HD Images https://www.onlin... R Title Icon (Favicon) Readymade Icon HTTP Status 404 ...
Microsoft Azure Search resources, services, and docs (G+) Ranjithbhowmick4027@... DEFAULT DIRECTORY

Home > API Management services > WeatherForecastManagement

API Management service

WeatherForecastManagement | OAuth 2.0 + OpenID Connect

OAuth 2.0 OpenID Connect

OAuth 2.0

+ Add Columns Refresh

Search to filter items...

| Display name |
|--|
| Unique name used to reference this authorization server on the client application. |

Add OAuth2 service

API Management service

Display name *

Id *

Description

Authorization server description

Client registration page URL *

Client registration endpoint is used for registering clients with the authorization server.

Authorization grant types

Authorization code

Implicit

Resource owner password

Client credentials

Authorization endpoint URL *

Authorization endpoint is used to authenticate resource owners.

Support state parameter

Authorization request method

GET

POST

Create

Filter for any field...

Name ↑

WeatherForecastManagement

Developer portal

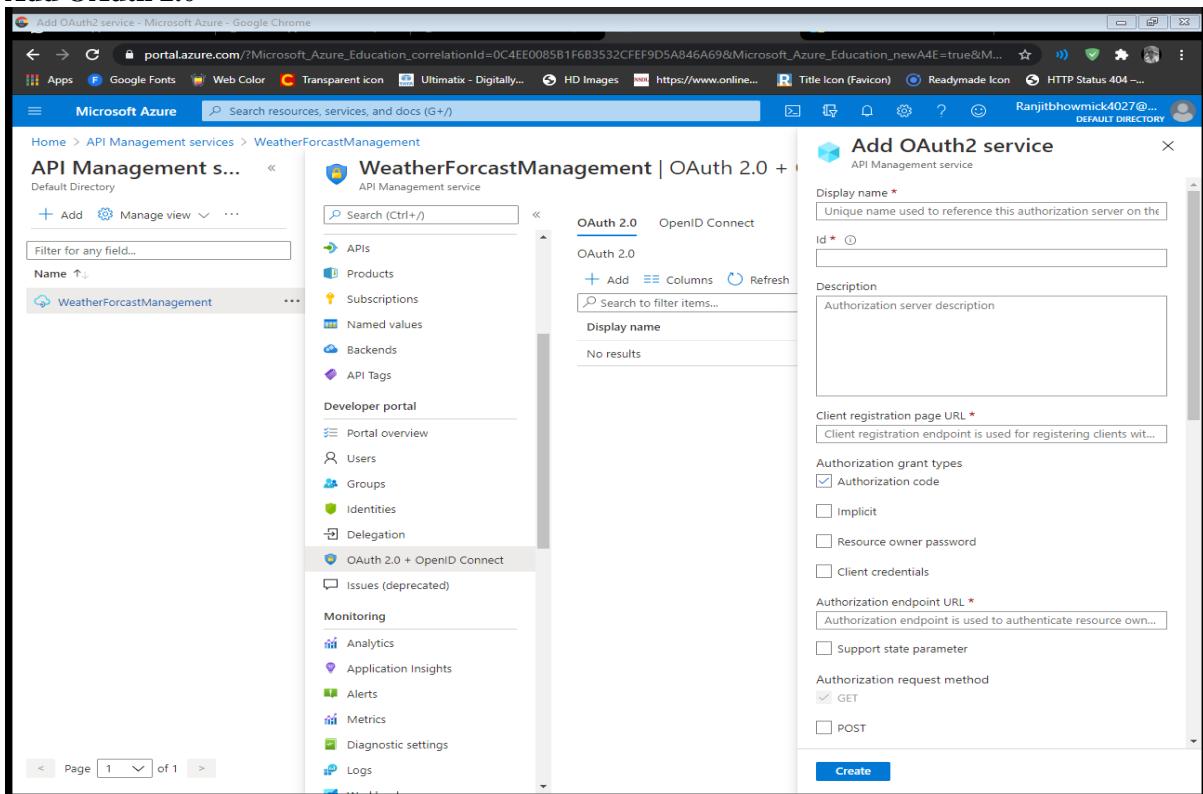
- Portal overview
- Users
- Groups
- Identities
- Delegation
- OAuth 2.0 + OpenID Connect**
- Issues (deprecated)

Monitoring

- Analytics
- Application Insights
- Alerts
- Metrics
- Diagnostic settings
- Logs

Add Manage view ...

Page 1 of 1



Enter all details as below –

The screenshot shows two consecutive steps in the Azure API Management portal for creating an OAuth 2.0 service.

Step 1: Initial Configuration

The left pane shows the API Management service "WeatherForecastManagement". The right pane displays the "Add OAuth2 service" configuration page for "OAuth 2.0".

- Display name:** KJ-RANJIT-WeatherForecastAuth
- Id:** kj-ranjit-weatherforecastauth
- Description:** KJ-RANJIT-WeatherForecastAuth 2.0
- Client registration page URL:** http://localhost
- Authorization grant types:** Authorization code (selected)
- Authorization endpoint URL:** https://login.microsoftonline.com/5768db3a-7236-4687-a...
- Support state parameter:** (unchecked)
- Authorization request method:** GET (selected)

Create button is visible at the bottom.

Step 2: Advanced Configuration

The configuration has been expanded to show more detailed settings.

- Authorization request method:** POST (selected)
- Token endpoint URL:** https://login.microsoftonline.com/5768db3a-7236-4687-a...
- Additional body parameters using application/x-www-form-urlencoded format:** A table with one row: Name (No results) and Value (No results).
- Client authentication methods:** Basic (unchecked), In the body (selected)
- Access token sending method:** Authorization header (selected)
- Query parameters:** (unchecked)
- Default scope:** Authorization server default scope
- Client credentials:**
 - Client ID:** 52b3bf4f-c1e4-4aba-b243-48650be2d6e5
 - Client secret:** 5ce7ea6d-c281-4079-9785-9c1c259b5b26

Create button is visible at the bottom.

The screenshot shows the Microsoft Azure API Management service interface. On the left, the navigation pane includes links for Apps, Google Fonts, Web Color, Transparent Icon, Ultimatrix - Digitally..., HD Images, https://www.online..., Title Icon (Favicon), Readymade Icon, and HTTP Status 404 ...

The main content area displays the "WeatherForcastManagement | OAuth 2.0 + OpenID Connect" configuration page. The left sidebar lists various management options like APIs, Products, Subscriptions, Named values, Backends, and API Tags. The "OAuth 2.0 + OpenID Connect" section is currently selected.

The right side of the screen shows the configuration details for the OAuth 2.0 service. It includes fields for "Display name" (set to "WeatherForcastManagement"), "Client credentials" (Client ID: 52b3bf4f-c1e4-4aba-b243-48650be2d6e5, Client secret: 5ce7ea6d-c281-4079-9785-91c1c259b5b26), "Redirect URI" (https://weatherforcastmanagement.developer.azure-api....), and "Resource owner password credentials" (Resource owner username and Resource owner password fields).

At the bottom right, there is a "Create" button.

Click on create button –

The screenshot shows the Microsoft Azure portal interface for managing an API Management service named "WeatherForecastManagement". The left sidebar lists various management options like APIs, Products, Subscriptions, etc. The main content area displays the "OAuth 2.0 + OpenID Connect" configuration for this service. A success message at the top right states: "OAuth2 service added" and "OAuth2 service 'KJ-RANJIT-WeatherForecastAuth' was successfully added". The "OAuth 2.0" tab is selected, showing a table with one row:

| Display name | Description | ... |
|-------------------------------|-----------------------------------|-----|
| KJ-RANJIT-WeatherForecastAuth | KJ-RANJIT-WeatherForecastAuth 2.0 | ... |

Now navigate to Client App - KJ-WeatherForecastClient-

The screenshot shows the Microsoft Azure portal interface for managing a client application named "KJ-WeatherForecastClient".

Left Sidebar:

- Overview
- Quickstart
- Integration assistant
- Authentication** (selected)
- Branding
- Certificates & secrets
- Token configuration
- API permissions
- Expose an API
- App roles | Preview
- Owners
- Roles and administrators | Preview
- Manifest

Support + Troubleshooting:

- Troubleshooting
- New support request

Main Content Area:

KJ-WeatherForecastClient | Authentication

Search (Ctrl+ /) Save Discard Got feedback?

Depending on the platform or device this application is targeting, additional configuration may be required such as redirect URIs, specific authentication settings, or fields specific to the platform.

Add a platform

Supported account types

Who can use this application or access this API?

Accounts in this organizational directory only (Default Directory only - Single tenant)
 Accounts in any organizational directory (Any Azure AD directory - Multitenant)

Help me decide...

Warning: Due to temporary differences in supported functionality, we don't recommend enabling personal Microsoft accounts for an existing registration. If you need to enable personal accounts, you can do so using the manifest editor. [Learn more about these restrictions.](#)

Warning: Starting November 9th, 2020 end users will no longer be able to grant consent to newly registered multitenant apps without verified publishers. Add MPN ID to verify publisher

Advanced settings

Allow public client flows

Enable the following mobile and desktop flows:

App collects plaintext password (Resource Owner Password Credential Flow) [Learn more](#)
 No keyboard (Device Code Flow) [Learn more](#)
 SSO for domain-joined Windows (Windows Integrated Auth Flow) [Learn more](#)

Now add a platform as web –

The screenshot shows the Microsoft Azure portal interface. On the left, the navigation menu is visible with sections like Home, Overview, Quickstart, Integration assistant, Manage, and Support + Troubleshooting. Under Manage, the Authentication section is selected. In the center, the main content area displays the 'Configure platforms' dialog. This dialog is divided into several sections:

- Web applications:** Contains two items: 'Web' (selected) and 'Single-page application'. The 'Web' item is described as "Build, host, and deploy a web server application. .NET, Java, Python".
- Mobile and desktop applications:** Contains two items: 'iOS / macOS' and 'Android'. The 'iOS / macOS' item is described as "Objective-C, Swift, Xamarin".
- Mobile and desktop applications:** Contains one item: 'Mobile and desktop applications'. It is described as "Windows, UWP, Console, IoT & Limited-entry Devices, Classic iOS + Android".

Below these sections, there are several informational and warning messages:

- A message about supported account types: "Who can use this application or access this API? Accounts in this organizational directory only (Default Directory on tenant) or Accounts in any organizational directory (Any Azure AD directory - multi-tenant). Help me decide..."
- A warning message: "⚠ Due to temporary differences in supported functionality, we don't recommend accounts for an existing registration. If you need to enable personal accounts, use the editor. Learn more about these restrictions."
- A warning message: "⚠ Starting November 9th, 2020 end users will no longer be able to grant apps without verified publishers. Add MPN ID to verify publisher"

At the bottom, there are sections for "Advanced settings" (with a link to "Allow public client flows") and "Enable the following mobile and desktop flows:" (with a list of three options: "App collects plaintext password (Resource Owner Password Credential)", "No keyboard (Device Code Flow)", and "SSO for domain-joined Windows (Windows Integrated Auth Flow)").

Paste the Redirect URL and click on configure-

The screenshot shows the Microsoft Azure 'Configure Web' dialog box. On the left, there's a sidebar with links like Overview, Quickstart, Integration assistant, Manage (Branding, Authentication, Certificates & secrets, Token configuration, API permissions, Expose an API, App roles | Preview, Owners, Roles and administrators | Preview, Manifest), Support + Troubleshooting (Troubleshooting, New support request), and a 'Help me decide...' button. The main area is titled 'Configure Web' and has tabs for All platforms, Quickstart, and Docs. Under 'Redirect URLs', it says: 'The URIs we will accept as destinations when returning authentication responses (tokens) after successfully authenticating or signing out users. Also referred to as reply URLs.' A text input field contains the URL 'https://weatherforecastmanagement.developer.azure-api.net/signin-oauth/code/callback'. Below this, under 'Front-channel logout URL', there's a note about clearing session data and a text input field with 'e.g. https://example.com/logout'. Under 'Implicit grant and hybrid flows', there's a note about tokens and a checkbox for 'Access tokens (used for implicit flows)'. At the bottom right are 'Configure' and 'Cancel' buttons.

Page will look like this –

The screenshot shows the Microsoft Azure portal interface. The URL in the address bar is `portal.azure.com/?Microsoft_Azure_Education_correlationId=0C4EE0085B1F6B3532CFEF9D5A846A69&Microsoft_Azure_Education_newA4E=true&M...`. The title bar says "KJ-WeatherForecastClient - Microsoft Azure - Google Chrome". The main navigation bar has "Microsoft Azure" selected. Below it, the breadcrumb navigation shows "Home > KJ-WeatherForecastClient". The page title is "KJ-WeatherForecastClient | Authentication". On the left, there's a sidebar with "Overview", "Quickstart", "Integration assistant", "Manage" (selected), "Branding", "Authentication" (selected), "Certificates & secrets", "Token configuration", "API permissions", "Expose an API", "App roles | Preview", "Owners", "Roles and administrators | Preview", "Manifest", "Support + Troubleshooting" (selected), "Troubleshooting", and "New support request". The main content area is titled "Platform configurations" and "Web". It shows "Redirect URIs" with one entry: `https://weatherforecastmanagement.developer.azure-api.net/signin-oauth/code/callback/kj-ranjit-weatherforecastauth`. There's also a "Front-channel logout URL" input field with the placeholder "e.g. https://example.com/logout". At the bottom, there's a section for "Implicit grant and hybrid flows" with two checkboxes: "Access tokens (used for implicit flows)" and "ID tokens (used for implicit and hybrid flows)".

Now to enable OAuth 2.0 user authorization for your API –

Browser API Management service and select your service-

API Management services - Microsoft Azure - Google Chrome
portal.azure.com/?Microsoft_Azure_Education_correlationId=0C4EE0085B1F6B3532CFFEF9D5A846A69&Microsoft_Azure_Education_newA4E=true&M...
Apps Google Fonts Web Color Transparent icon Ultimatrix - Digitally... HD Images https://www.onlin... Title Icon (Favicon) Readymade Icon HTTP Status 404 ...
Microsoft Azure Search resources, services, and docs (G+)
Home >
API Management services
Default Directory
Add Manage view Refresh Export to CSV Open query Assign tags Feedback
Filter for any field... Subscription == all Resource group == all Location == all Add filter
Showing 1 to 1 of 1 records.

| Name | Status | Tier | Type | Location | Resource group | Subscription |
|--------------------------|--------|-----------|------------------------|----------|--------------------------|--------------------|
| WeatherForcastManagement | Online | Developer | API Management service | East US | WeatherForcastManagement | Azure for Students |

No grouping List view
< Previous Page 1 of 1 Next >

Select API

WeatherForcastManagement - Microsoft Azure - Google Chrome
portal.azure.com/?Microsoft_Azure_Education_correlationId=0C4EE0085B1F6B3532CFFEF9D5A846A69&Microsoft_Azure_Education_newA4E=true&M...
Apps Google Fonts Web Color Transparent icon Ultimatrix - Digitally... HD Images https://www.onlin... Title Icon (Favicon) Readymade Icon HTTP Status 404 ...
Microsoft Azure Search resources, services, and docs (G+)
Home > API Management services <
API Management services
Default Directory
Add Manage view ...
Filter for any field...
Name WeatherForcastManagement

WeatherForcastManagement API Management service
Overview Developer portal Delete Share to mobile
Resource group (change) WeatherForcastManagement
Status Online
Location East US
Subscription (change) Azure for Students
Subscription ID bcb994c8-b7b8-4253-b727-d074869e784a
Tags (change) Click here to add tags
Properties Get started Learn more Monitor Recommendations (0)

Pricing tier
Pricing tier Developer
SLA No
Scale 1 unit

External cache
External cache None

Virtual network
Virtual network None

Custom domains
Gateway URL https://weatherforcastmanagement.azure-api.net
View all
Page 1 of 1

Select “echo api” – any api you want to protect –

The screenshot shows the Microsoft Azure API Management service interface. On the left, there's a sidebar with various navigation options like Overview, Activity log, Access control (IAM), Tags, Diagnose and solve problems, Properties, Locks, APIs, Products, Subscriptions, Named values, Backends, API Tags, Portal overview, Users, Groups, Identities, and Delegation. The main area is titled "WeatherForcastManagement | APIs" and shows a list of APIs. One API, "Echo API", is highlighted with a blue background. At the top right, there are tabs for Design, Settings, Test, Revisions, and Change log. Below the tabs, there are sections for "Operations" and "Definitions". The URL in the browser bar is https://portal.azure.com/#Microsoft_Azure_Education_correlationId=0C4EE0085B1F6B3532CFEF9D5A846A69&Microsoft_Azure_Education_newA4E=true&M....

Now move to setting and Select OAUTH 2.0 which have been configured –

The screenshot shows the Azure API Management service interface for the 'WeatherForecastManagement' API. The left sidebar lists 'All APIs' with 'Echo API' selected. The main content area displays 'REVISION 1' (CREATED Jan 30, 2021, 5:34:12 PM) with tabs for Design, Settings, Test, Revisions, and Change log. The 'Test' tab is active. The 'Subscription' section has 'Subscription required' checked and 'Header name' set to 'Ocp-Apim-Subscription-Key'. The 'Security' section shows 'User authorization' set to 'OAuth 2.0' with 'OAuth 2.0 server' set to 'KJ-RANJIT-WeatherForecastAuth'. The 'Diagnostics Logs' section has 'Application Insights' selected. At the bottom are 'Save' and 'Discard' buttons.

Now click on Save-

WeatherForecastManagement - Microsoft Azure - Google Chrome

portal.azure.com/?Microsoft_Azure_Education_correlationId=0C4EE0085B1F6B3532CFEF9D5A846A69&Microsoft_Azure_Education_newA4E=true&M...

Apps Google Fonts Web Color Transparent icon Ultimatrix - Digitally... HD Images https://www.onlin... Title Icon (Favicon) Readymade Icon HTTP Status 404 ...

Microsoft Azure Search resources, services, and docs (G+) Ranjithbhowmick4027@... DEFAULT DIRECTORY

Home > API Management services > WeatherForecastManagement

WeatherForecastManagement | APIs API Management service

Developer portal

REVISION 1 CREATED Jan 30, 2021, 5:34:12 PM

Design Settings Test Revisions Change log

Subscription

Subscription required

Header name Ocp-Apim-Subscription-Key

Query parameter name subscription-key

Security

User authorization None OAuth 2.0 OpenID connect

OAuth 2.0 server KJ-RANJIT-WeatherForecastAuth

Override scope

Diagnostics Logs

Application Insights Azure Monitor

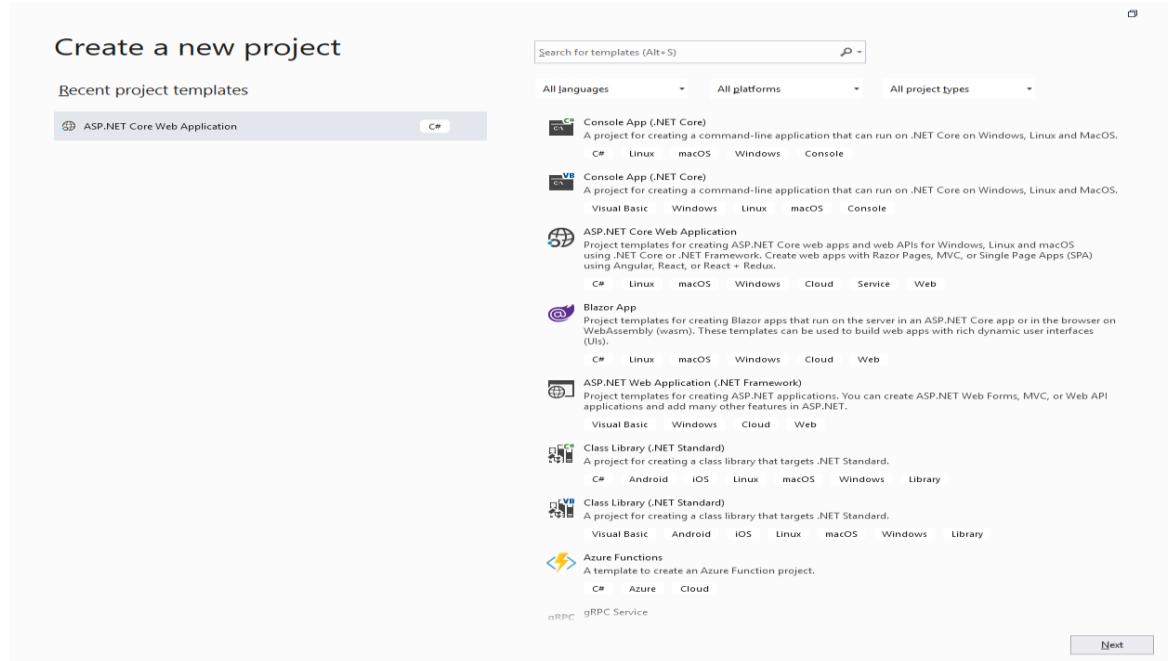
Enable

API Management API "Echo API" saved successfully. 6:37 PM

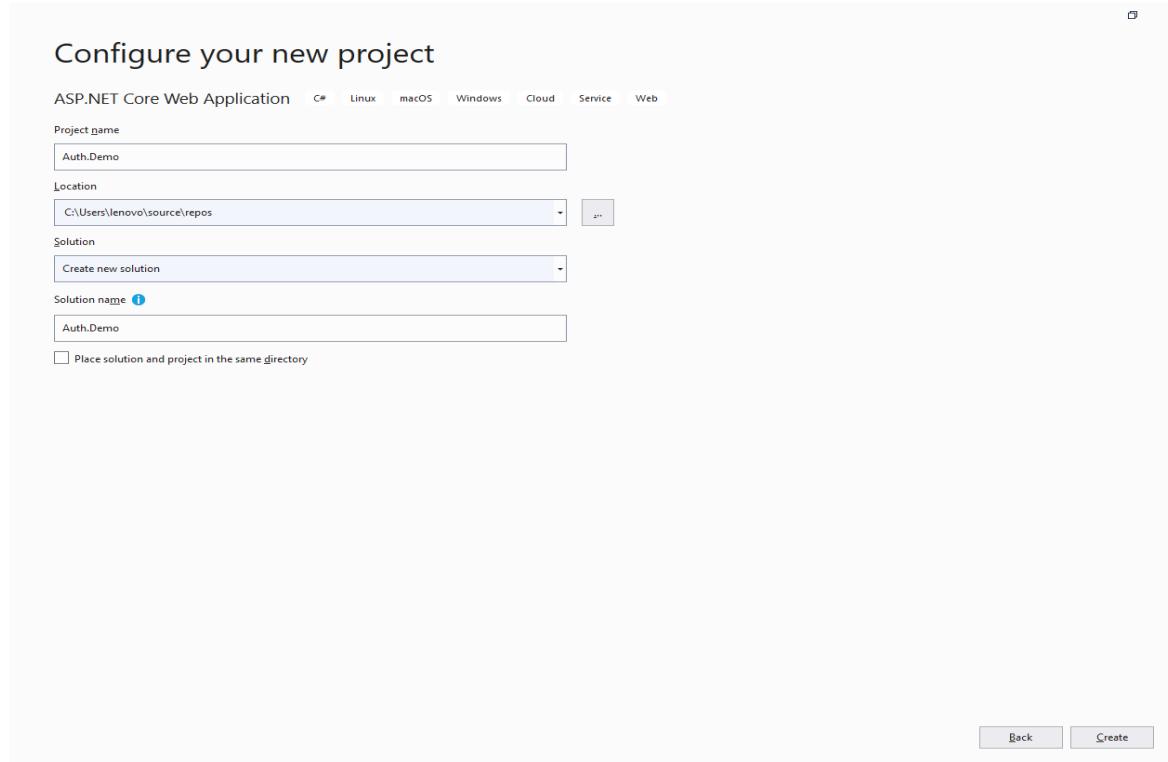
The screenshot shows the Azure API Management interface for the 'WeatherForecastManagement' service. On the left, there's a sidebar with search and filter options, and a list of APIs including 'Echo API'. The main area is titled 'WeatherForecastManagement | APIs' and shows 'REVISION 1' created on Jan 30, 2021. It has tabs for Design, Settings, Test, Revisions, and Change log. Under 'Subscription', 'Subscription required' is checked. Under 'Security', 'User authorization' is set to 'OAuth 2.0' with 'KJ-RANJIT-WeatherForecastAuth' selected. Under 'Diagnostics Logs', 'Application Insights' is chosen. A success message 'API "Echo API" saved successfully.' is shown at the top right.

Issuing a custom JWT token using a symmetric signing key

Create new Asp.Net Core Project –



Enter Details and Select API Option –



Follow below steps –

Create Startup.cs and use below Code –

```
using Microsoft.AspNetCore.Authentication.JwtBearer;
using Microsoft.AspNetCore.Builder;
using Microsoft.AspNetCore.Hosting;
using Microsoft.AspNetCore.Mvc;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace Auth.Demo
{
    public class Startup
    {
        public Startup(IConfiguration configuration)
        {
            Configuration = configuration;
        }

        public IConfiguration Configuration { get; }

        // This method gets called by the runtime. Use this method to add services to the
        // container.
        public void ConfigureServices(IServiceCollection services)
        {
            services.AddControllers();
            var key = "This is my test key";

            services.AddAuthentication(x =>
            {
                x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
                x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
            }).AddJwtBearer(x =>
            {
                x.RequireHttpsMetadata = false;
                x.SaveToken = true;
                x.TokenValidationParameters = new
                    Microsoft.IdentityModel.Tokens.TokenValidationParameters
                {
                    ValidateIssuerSigningKey = true,
                    IssuerSigningKey = new
                        SymmetricSecurityKey(System.Text.Encoding.ASCII.GetBytes(key)),
                    ValidateIssuer = false,
                    ValidateAudience = false
                };
            });
        }
    }
}
```

```
});

        services.AddSingleton<IJWTAuthenticationManager>(new
JwtAuthenticationManager(key));

}

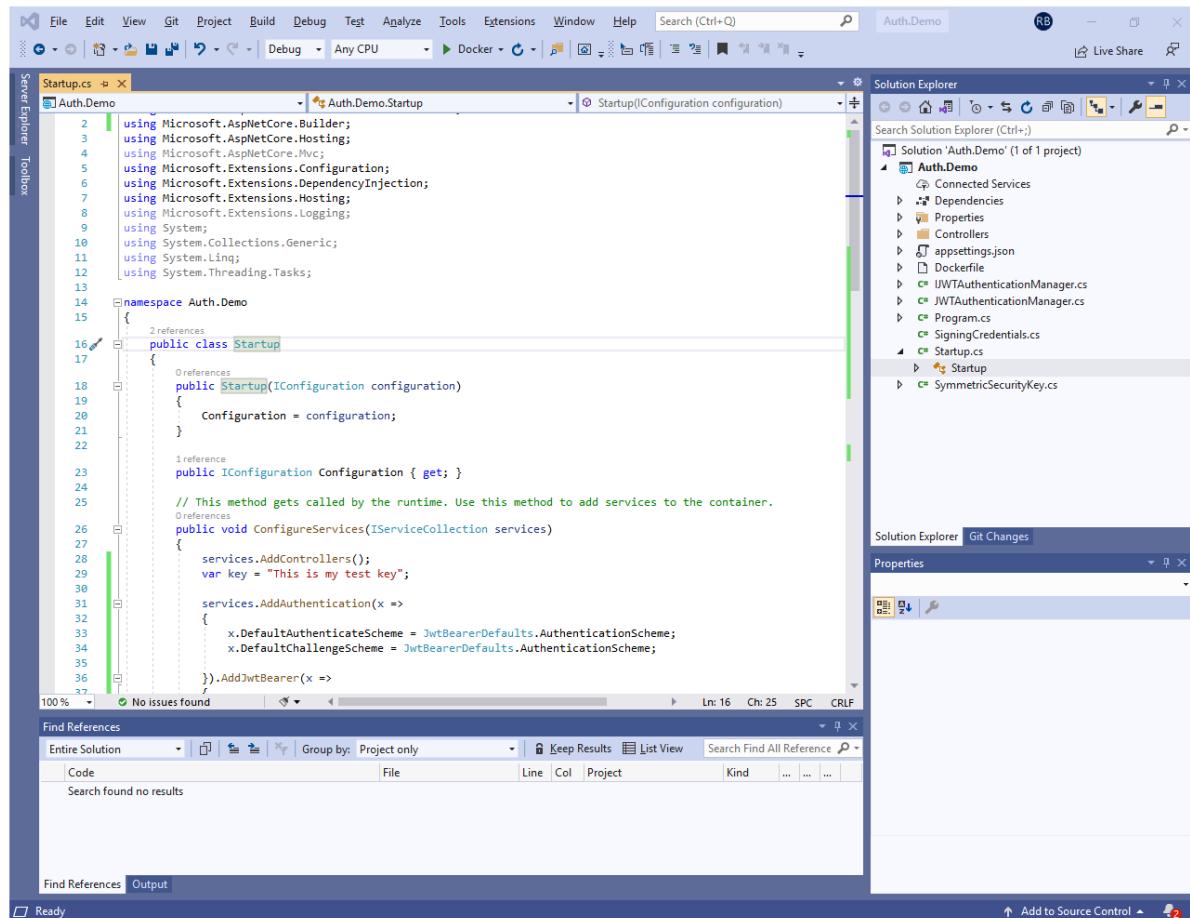
// This method gets called by the runtime. Use this method to configure the HTTP
request pipeline.
public void Configure(IApplicationBuilder app, IWebHostEnvironment env)
{
    if (env.IsDevelopment())
    {
        app.UseDeveloperExceptionPage();
    }

    app.UseRouting();

    app.UseAuthentication();

    app.UseAuthorization();

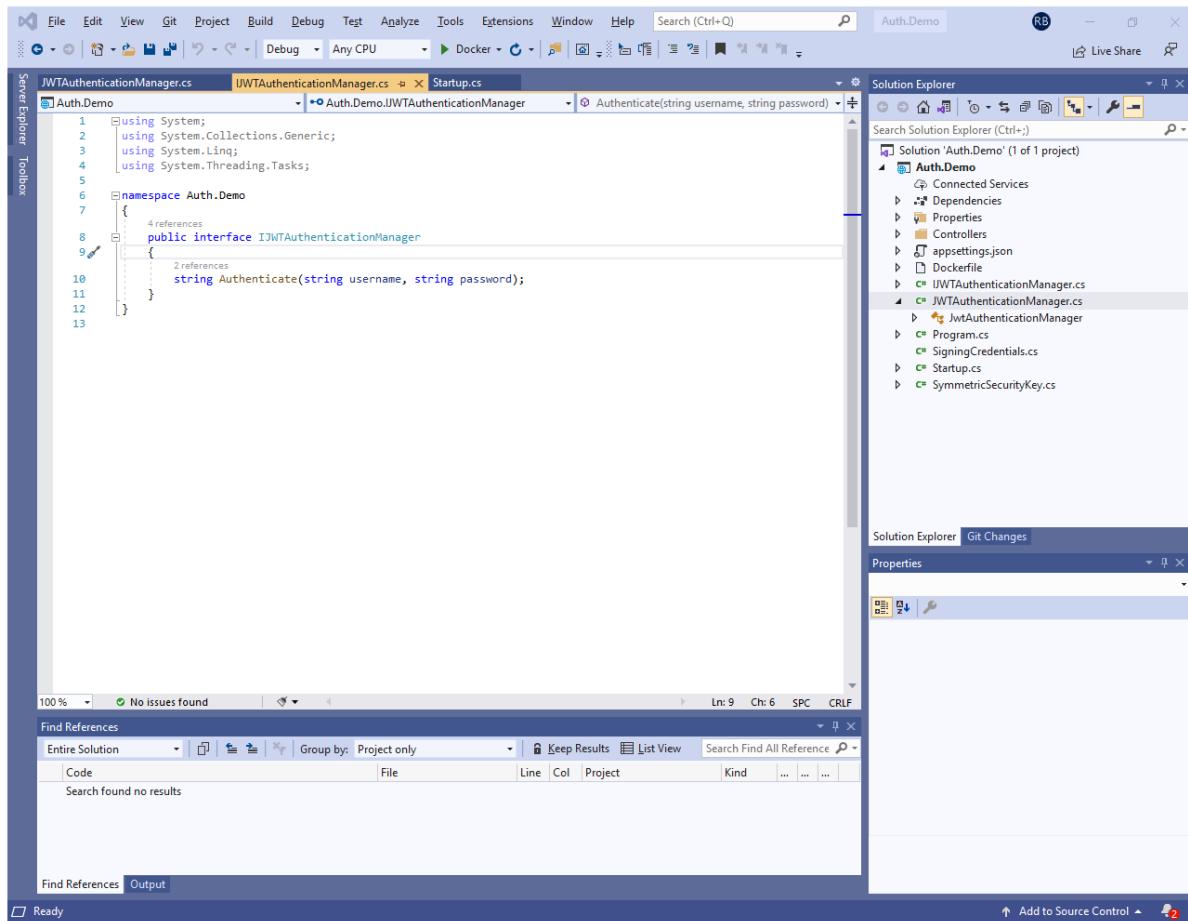
    app.UseEndpoints(endpoints =>
    {
        endpoints.MapControllers();
    });
}
}
```



Create Interface - IJWTAuthenticationManager.cs

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace Auth.Demo
{
    public interface IJWTAuthenticationManager
    {
        string Authenticate(string username, string password);
    }
}
```



```
using Microsoft.IdentityModel.Tokens;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Security.Claims;
using System.Threading.Tasks;

namespace Auth.Demo
{
    public class JwtAuthenticationManager : IJWTAuthenticationManager
    {
        private readonly IDictionary<string, string> users = new Dictionary<string, string>
        { { "test1", "password1"}, { "test2", "password2"} };
        private readonly string key;

        public JwtAuthenticationManager(string key)
        {
            this.key = key;
        }

        public string Authenticate(string username, string password)
        {
            if (!users.Any(u => u.Key == username && u.Value == password))

```

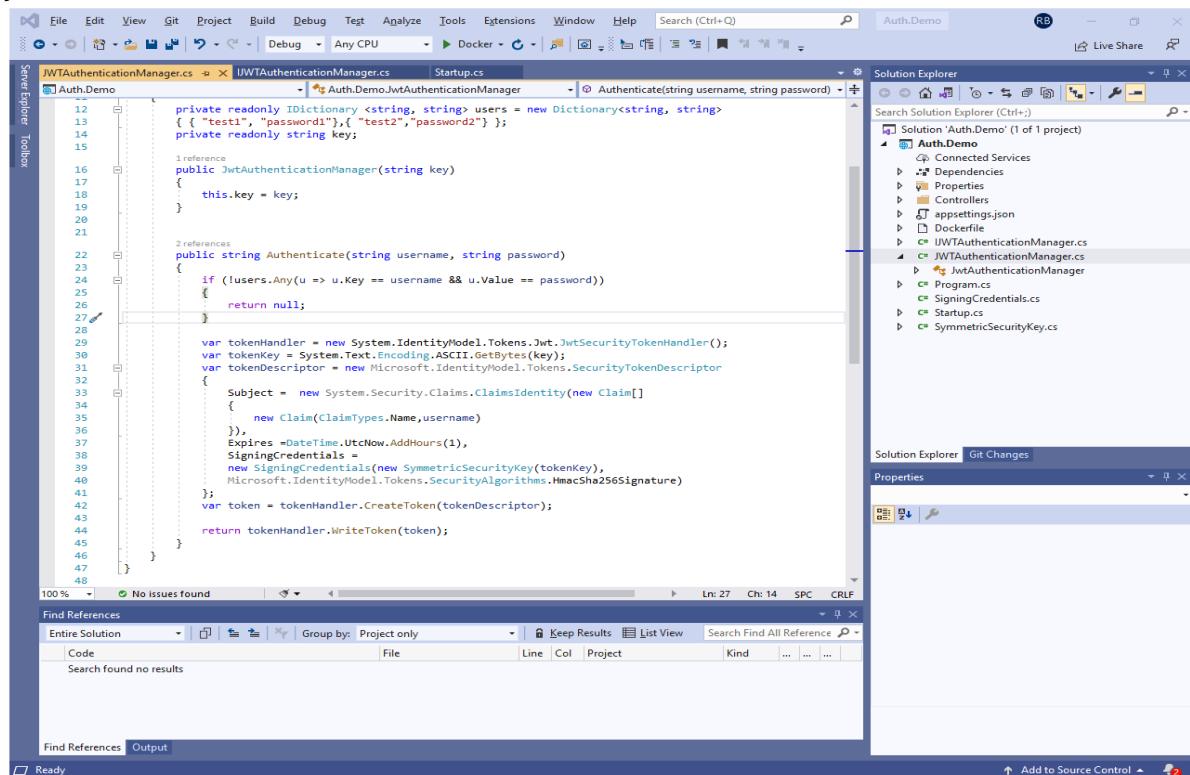
```

    {
        return null;
    }

    var tokenHandler = new
System.IdentityModel.Tokens.Jwt.JwtSecurityTokenHandler();
    var tokenKey = System.Text.Encoding.ASCII.GetBytes(key);
    var tokenDescriptor = new
Microsoft.IdentityModel.Tokens.SecurityTokenDescriptor
{
    Subject = new System.Security.Claims.ClaimsIdentity(new Claim[]
    {
        new Claim(ClaimTypes.Name,username)
    }),
    Expires =DateTime.UtcNow.AddHours(1),
    SigningCredentials =
    new SigningCredentials(new SymmetricSecurityKey(tokenKey),
    Microsoft.IdentityModel.Tokens.SecurityAlgorithms.HmacSha256Signature)
};
    var token = tokenHandler.CreateToken(tokenDescriptor);

    return tokenHandler.WriteToken(token);
}
}
}
}

```



Create NameController.cs and follow below code –

```

using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using System;
using System.Collections.Generic;
using System.Linq;

```

```
using System.Threading.Tasks;

// For more information on enabling Web API for empty projects, visit
// https://go.microsoft.com/fwlink/?LinkID=397860

namespace Auth.Demo.Controllers
{
    [Authorize]
    [Route("api/[controller]")]
    [ApiController]
    public class NameController : ControllerBase
    {
        private readonly IJWTAuthenticationManager jWTAuthenticationManager;

        public NameController(IJWTAuthenticationManager jWTAuthenticationManager)
        {
            this.jWTAuthenticationManager = jWTAuthenticationManager;
        }

        // GET: api/<NameController>
        [HttpGet]
        public IEnumerable<string> Get()
        {
            return new string[] { "KURLA", "ULHASNAGAR" };
        }

        // GET api/<NameController>/5
        [HttpGet("{id}")]
        public string Get(int id)
        {
            return "value";
        }

        [AllowAnonymous]
        [HttpPost("authenticate")]
        public IActionResult Authenticate([FromBody] UserCred userCred)
        {
            var token = jWTAuthenticationManager.Authenticate(userCred.Username,
userCred.Password);
            if (token == null)
                return Unauthorized();

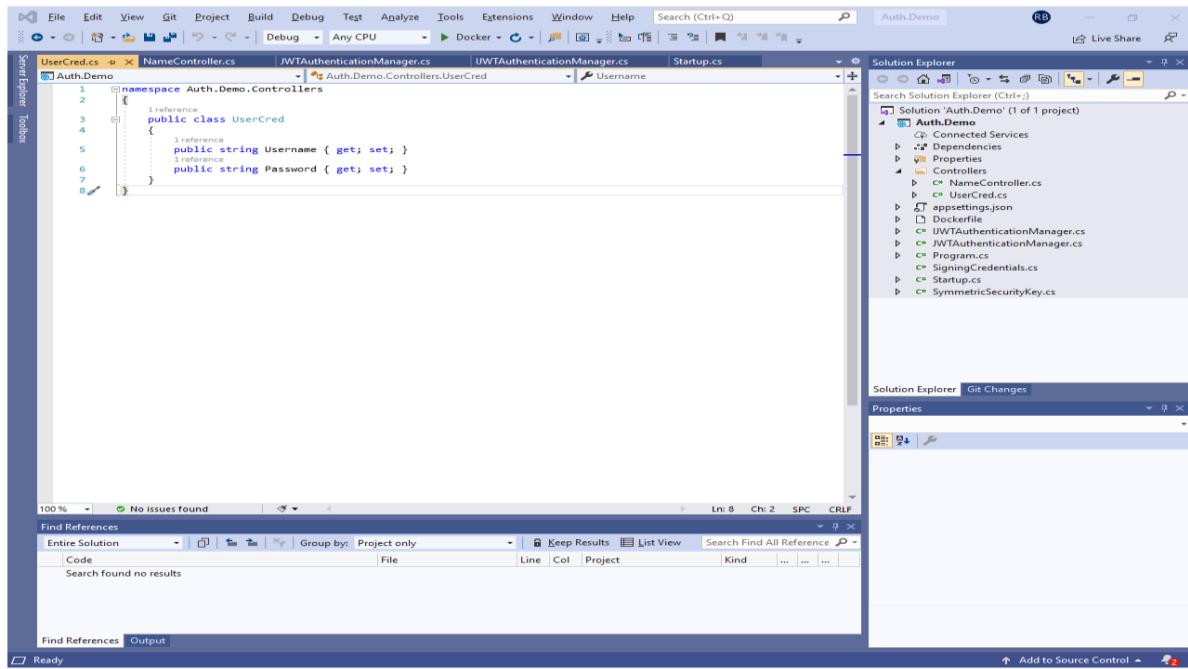
            return Ok(token);
        }
    }
}
```

The screenshot shows the Visual Studio IDE interface with the following details:

- File Menu:** File, Edit, View, Git, Project, Build, Debug, Test, Analyze, Tools, Extensions, Window, Help.
- Solution Explorer:** Shows the 'Auth.Demo' solution with one project 'Auth.Demo'. The project contains files: Connected Services, Dependencies, Properties, Controllers (containing NameController.cs, UserCred.cs), Dockerfile, appsettings.json, IWTAuthenticationManager.cs, JWTAuthenticationManager.cs, Program.cs, SigningCredentials.cs, Startup.cs, and SymmetricSecurityKey.cs.
- Code Editor:** Displays the 'NameController.cs' file. The code defines a controller that returns an array of strings ('KURLA', 'ULHASNAGAR') via GET requests.
- Toolbox:** Standard Visual Studio toolbox items.
- Find References:** Shows results for 'Entire Solution' with no results found.
- Status Bar:** Lr: 21 Ch: 51 SPC: CRLF.

Create UserCred.cs and follow below code –

```
namespace Auth.Demo.Controllers
{
    public class UserCred
    {
        public string Username { get; set; }
        public string Password { get; set; }
    }
}
```



Keep Program.cs as it is-

```

using Microsoft.AspNetCore.Hosting;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.Hosting;
using Microsoft.Extensions.Logging;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;

namespace Auth.Demo
{
    public class Program
    {
        public static void Main(string[] args)
        {
            CreateHostBuilder(args).Build().Run();
        }

        public static IHostBuilder CreateHostBuilder(string[] args) =>
            Host.CreateDefaultBuilder(args)
                .ConfigureWebHostDefaults(webBuilder =>
                {
                    webBuilder.UseStartup<Startup>();
                });
    }
}

```

The screenshot shows the Microsoft Visual Studio IDE interface. The main window displays the code editor with the file `Program.cs` open. The code implements a `Program` class with a `Main` method that creates a host builder and runs it. The `Find References` dialog is open at the bottom, showing results for the entire solution, with no results found.

```
1  using Microsoft.AspNetCore.Hosting;
2  using Microsoft.Extensions.Configuration;
3  using Microsoft.Extensions.Hosting;
4  using Microsoft.Extensions.Logging;
5  using System;
6  using System.Collections.Generic;
7  using System.Linq;
8  using System.Threading.Tasks;
9
10 namespace Auth.Demo
11 {
12     public class Program
13     {
14         public static void Main(string[] args)
15         {
16             CreateHostBuilder(args).Build().Run();
17         }
18
19         public static IHostBuilder CreateHostBuilder(string[] args) =>
20             Host.CreateDefaultBuilder(args)
21                 .ConfigureWebHostDefaults(webBuilder =>
22                 {
23                     webBuilder.UseStartup<Startup>();
24                 });
25
26     }
27 }
```

In next step, getting error as below –

The screenshot shows the Microsoft Visual Studio IDE interface with the following components visible:

- Code Editor:** The main window displays the `Startup.cs` file for the project `Auth.Demo`. The code implements the `IConfiguration` interface and configures services using `IServiceCollection`.
- Solution Explorer:** Shows the project structure for `Auth.Demo`, which includes files like `NameController.cs`, `UserCred.cs`, `appsettings.json`, `Dockerfile`, `JWTAuthenticationManager.cs`, `JWTAuthenticationManager.cs`, `Program.cs`, `SigningCredentials.cs`, and `Startup.cs`.
- Error List:** The bottom-left pane shows one error: `CTC1010` - The current user is not in the 'docker-users' group. Add yourself to the 'docker-users' group and then log out and back in to Windows.
- Properties:** The bottom-right pane shows the properties for the selected item in the Solution Explorer.

```
18 public Startup(IConfiguration configuration)
19 {
20     Configuration = configuration;
21 }
22
23 [reference]
24 public IConfiguration Configuration { get; }
25
26 // This method gets called by the runtime. Use this method to add services to the container.
27 [reference]
28 public void ConfigureServices(IServiceCollection services)
29 {
30     services.AddControllers();
31     var key = "This is my test key";
32
33     services.AddAuthentication(x =>
34     {
35         x.DefaultAuthenticateScheme = JwtBearerDefaults.AuthenticationScheme;
36         x.DefaultChallengeScheme = JwtBearerDefaults.AuthenticationScheme;
37
38         x.AddJwtBearer(x =>
39         {
40             x.RequireHttpsMetadata = false;
41             x.SaveToken = true;
42             x.TokenValidationParameters = new Microsoft.IdentityModel.Tokens.TokenValidationParameters
43             {
44                 ValidateIssuerSigningKey = true,
45                 IssuerSigningKey = new Microsoft.IdentityModel.Tokens.SymmetricSecurityKey(System.Text.Encoding.UTF8.GetBytes(key)),
46                 ValidateIssuer = false,
47                 ValidateAudience = false
48             };
49         });
50     });
51 }
52
53
54 }
```

Steps need to follow once the above issue get fixed.

1. Now run the application and paste the URL in Postman-
2. Pass the data as below
3. Use data as –
- 4.

{“username”：“test1”,“password”：“password1” }

A token will get generated in encrypted form. Pass the Authorization key value with generated token.

Practical no 10

Create a serverless API using Azure functions

About Azure Function:



Azure Functions is a cloud service available on-demand that provides all the continually-updated infrastructure and resources needed to run your applications.

You focus on the pieces of code that matter most to you, and Azure Functions handles the rest.

Serverless = LESS Server (Your code will not run 24/7) Azure will take the code for API and save it to file.

It will not run until it gets triggered, i.e., Server will not work until someone is really using it.

Azure has many serverless offerings, like Azure Functions, Logic Apps, Azure Storage, Cloud messaging (Event Grid), Azure bot Services two major serverless offerings, Azure Functions and Logic Apps but as per current trend and market demand, we will be exploring Azure Functions and Logic Apps.

Azure Functions is a FaaS offering—developers can write custom logic based on available triggers and execute the logic. Logic Apps is a BaaS offering where different actions and workflow conditions are available to execute a logic based on an event trigger. Both Azure Functions and Logic Apps support HTTP events, meaning both can act as standalone APIs.

Please see below steps and screenshots to create serverless API using Azure function:

Go to Azure Portal:

[Home - Microsoft Azure](#)

Sign in with your Microsoft account to access various Azure functionalities

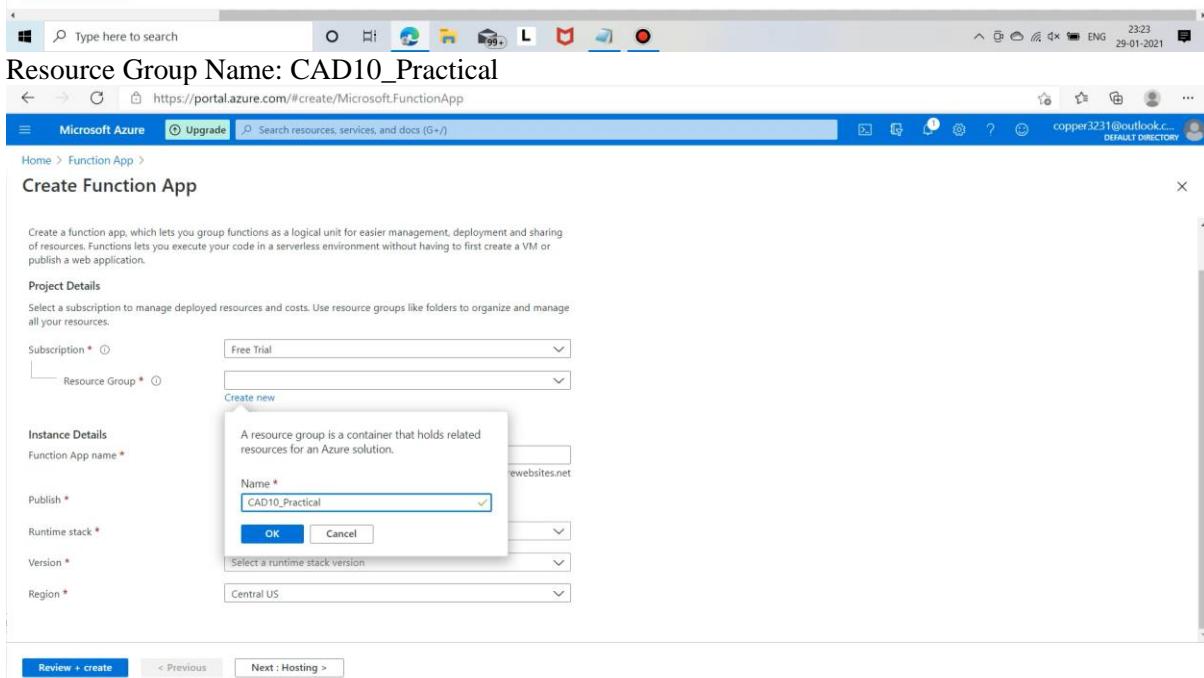
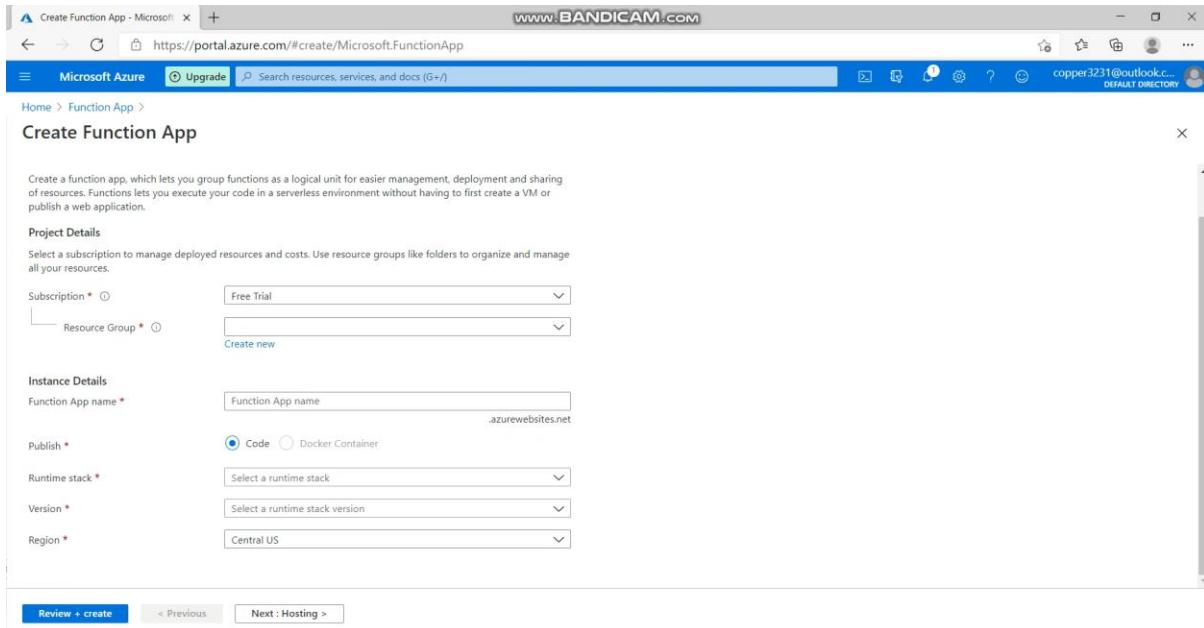
The screenshot shows the Microsoft Azure Home page. At the top, there's a navigation bar with icons for back, forward, search, and user information. Below it is the 'Azure services' navigation bar with icons for Create a resource, Virtual machines, App Services, Storage accounts (which is selected), SQL databases, Azure Database for PostgreSQL, Azure Cosmos DB, Kubernetes services, Function App, and More services. Under 'Navigate', there are links for Subscriptions, Resource groups, All resources, and Dashboard. In the 'Tools' section, there are links for Microsoft Learn, Azure Monitor, Security Center, and Cost Management. The 'Useful links' section includes Technical Documentation, Azure Services, Recent Azure Updates, and links to the App Store and Google Play for the Azure mobile app.

<https://portal.azure.com/#blade/HubsExtension/BrowseResourceBlade/resourceType/Microsoft.Web%2Fsites/functionapp>

After Successful sing in, Click on Function app → Click on Create Function App

The screenshot shows the 'Function App' blade in the Azure portal. The URL in the address bar is <https://portal.azure.com/#blade/HubsExtension/BrowseResourceBlade/resourceType/Microsoft.Web%2Fsites/functionapp>. The page displays a table with columns for Name, Status, Location, Pricing Tier, App Service Plan, Subscription, and App Type. A large lightning bolt icon is centered on the page, and the text 'No function apps to display' is shown. Below it, a message states 'The resources are currently filtered and not all resources may be displayed, such as hidden resources.' and 'Try changing your filters if you don't see what you're looking for.' There are buttons for 'Create Function App' and 'Clear filters / Show hidden'.

Click on Resource Group → Create New



Function App Name: CAD10_Practical_Sahil

https://portal.azure.com/#create/Microsoft.FunctionApp

Microsoft Azure Upgrade Search resources, services, and docs (G+/-)

Home > Function App > Create Function App

Create a function app, which lets you group functions as a logical unit for easier management, deployment and sharing of resources. Functions lets you execute your code in a serverless environment without having to first create a VM or publish a web application.

Project Details

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * Free Trial
 Resource Group * (New) CAD10_Practical
 Create new

Instance Details

Function App name * CAD10-Practical-Sahil
 .azurewebsites.net

Publish * Code Docker Container

Runtime stack * Select a runtime stack
 Version * Select a runtime stack version
 Region * Central US

Review + create < Previous Next : Hosting >

Runtime stack: .NET Core
 Version 3.1
 Region: Central US

Click on Review and Create:

https://portal.azure.com/#create/Microsoft.FunctionApp

Microsoft Azure Upgrade Search resources, services, and docs (G+/-)

Home > Function App > Create Function App

Create a function app, which lets you group functions as a logical unit for easier management, deployment and sharing of resources. Functions lets you execute your code in a serverless environment without having to first create a VM or publish a web application.

Project Details

Select a subscription to manage deployed resources and costs. Use resource groups like folders to organize and manage all your resources.

Subscription * Free Trial
 Resource Group * (New) CAD10_Practical
 Create new

Instance Details

Function App name * CAD10-Practical-Sahil
 .azurewebsites.net

Publish * Code Docker Container

Runtime stack * .NET Core
 Version * 3.1
 Region * Central US

Review + create < Previous Next : Hosting >

After review, click on Create:

← → ⌂ https://portal.azure.com/#create/Microsoft.FunctionApp

Microsoft Azure Upgrade Search resources, services, and docs (G+/-)

Home > Function App

Create Function App

Basics Hosting Monitoring Tags Review + create

Summary

 **Function App**
by Microsoft

Details

| | |
|----------------|--------------------------------------|
| Subscription | 52203fa7-a272-436a-bde5-27f391dbea0f |
| Resource Group | CAD10_Practical |
| Name | CAD10-Practical-Sahil |
| Runtime stack | .NET Core 3.1 |

Hosting

Plan (New)

| | |
|------------------|-------------------------|
| Plan type | App service plan |
| Name | ASP-CAD10Practical-9323 |
| Operating System | Windows |
| Region | Central US |
| SKU | Free |
| ACU | Shared infrastructure |

Create < Previous Next > Download a template for automation

Deployment will automatically start, please wait for it.

← → ⌂ https://portal.azure.com/#create/Microsoft.FunctionApp

Microsoft Azure Upgrade Search resources, services, and docs (G+/-)

Home > Function App

Create Function App

 **Function App**
by Microsoft

Details

| | |
|----------------|--------------------------------------|
| Subscription | 52203fa7-a272-436a-bde5-27f391dbea0f |
| Resource Group | CAD10_Practical |
| Name | CAD10-Practical-Sahil |
| Runtime stack | .NET Core 3.1 |

Hosting

Plan (New)

| | |
|------------------|-------------------------|
| Plan type | App service plan |
| Name | ASP-CAD10Practical-9323 |
| Operating System | Windows |
| Region | Central US |
| SKU | Free |
| ACU | Shared infrastructure |
| Memory | 1 GB memory |

Monitoring (New)

| | |
|----------------------|---------|
| Application Insights | Enabled |
|----------------------|---------|

*** Initializing deployment... 11:25 PM
Initializing template deployment to resource group 'CAD10_Practical'.

Create < Previous Next > Download a template for automation

The screenshot shows the Microsoft Azure portal interface for a deployment named "Microsoft.Web.FunctionApp-Portal-15a04307-b515". The main area displays deployment details, including the deployment name, subscription information (Free Trial), resource group (CAD10_Practical), start time (1/29/2021, 11:26:19 PM), and correlation ID (3226641b-6cb7-4226-8d41-cbf2961252fe). A message indicates "Deployment is in progress". On the left, there are navigation links for Overview, Inputs, Outputs, and Template. On the right, there are promotional links for Security Center, Free Microsoft tutorials, and Work with an expert.

Once the Deployment is complete, Click on Go to resource:

The screenshot shows the Microsoft Azure portal interface for the same deployment, now indicating "Your deployment is complete". The deployment status is marked with a green checkmark. The deployment details remain the same: deployment name, subscription, resource group, start time, and correlation ID. A success message at the top right states "Deployment succeeded" and "Deployment 'Microsoft.Web.FunctionApp-Portal-15a04307-b515' to resource group 'CAD10_Practical' was successful". On the right side, there are buttons for "Go to resource" and "Pin to dashboard".

Click on Functions → Add

Add function - Microsoft Azure

Microsoft Azure

www.BANDICAM.com

Home > Microsoft.Web.FunctionApp-Portal-15a04307-b515 > CAD10-Practical-Sahil

CAD10-Practical-Sahil | Functions

Function App

Search (Ctrl+ /)

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Security

Events (preview)

Functions

App keys

App files

Proxies

Deployment

Deployment slots

Deployment Center

Deployment Center (Preview)

Settings

Type here to search

23:29 29-01-2021

Click on HTTP Trigger and then click on Add:

Microsoft Azure

www.BANDICAM.com

Home > Microsoft.Web.FunctionApp-Portal-15a04307-b515 > CAD10-Practical-Sahil

CAD10-Practical-Sahil | Functions

Function App

Search (Ctrl+ /)

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Security

Events (preview)

Functions

App keys

App files

Proxies

Deployment

Deployment slots

Deployment Center

Deployment Center (Preview)

Settings

Type here to search

23:29 29-01-2021

Add function

Development environment Develop in portal

Select a template

Use a template to create a function. Triggers describe the type of events that invoke your functions. Learn more

Filter

| Template | Description |
|---------------------------------|--|
| HTTP trigger | A function that will be run whenever it receives an HTTP request, responding based on data in the body or query string |
| Timer trigger | A function that will be run on a specified schedule |
| Azure Queue Storage trigger | A function that will be run whenever a message is added to a specified Azure Storage queue |
| Azure Service Bus Queue trigger | A function that will be run whenever a message is added to a specified Service Bus queue |
| Azure Service Bus Topic trigger | A function that will be run whenever a message is added to the specified Service Bus topic |
| Azure Blob Storage trigger | A function that will be run whenever a blob is added to a |

Add Cancel

Microsoft Azure

www.BANDICAM.com

Home > Microsoft.Web.FunctionApp-Portal-15a04307-b515 > CAD10-Practical-Sahil

CAD10-Practical-Sahil | Functions

Function App

Search (Ctrl+ /)

Overview

Activity log

Access control (IAM)

Tags

Diagnose and solve problems

Security

Events (preview)

Functions

App keys

App files

Proxies

Deployment

Deployment slots

Deployment Center

Deployment Center (Preview)

Settings

Type here to search

11:29 PM

Creating function

Successfully created HttpTrigger1

Once the HTTP Trigger get created, you will see all the details about it.

Microsoft Azure Upgrade Search resources, services, and docs (G+)

Home > Microsoft.Web.FunctionApp-Portal-15a04307-b515 > CAD10-Practical-Sahil > HttpTrigger1

HttpTrigger1

Overview

Developer

Code + Test

Integration

Monitor

Function Keys

Search (Ctrl+ /)

Enable Disable Delete Get Function Url Refresh

Function app : CAD10-Practical-Sahil Status : Enabled Resource group (change) : CAD10_Practical Subscription (change) : Free Trial Subscription ID : 52203fa7-a272-436a-bde5-27f391dbea0f Application Insights : CAD10-Practical-Sahil JSON View

Total Execution Count

Successful Execution Count

Click on Code + Test, and update it as required

Microsoft Azure Upgrade Search resources, services, and docs (G+)

Home > Microsoft.Web.FunctionApp-Portal-15a04307-b515 > CAD10-Practical-Sahil > HttpTrigger1

HttpTrigger1 | Code + Test

Code + Test

Save Discard Refresh Test/Run Upload Get function URL

CAD10-Practical-Sahil \ HttpTrigger1 \ run.csx

```
1 #r "Newtonsoft.Json"
2
3 using System;
4 using Microsoft.AspNetCore.Mvc;
5 using Microsoft.Extensions.Primitives;
6 using Newtonsoft.Json;
7
8 public static async Task<ActionResult> Run(HttpContext req, ILogger log)
9 {
10     log.LogInformation("C# HTTP trigger function processed a request.");
11
12     string name = req.Query["name"];
13
14     string requestBody = await new StreamReader(req.Body).ReadToEndAsync();
15     dynamic data = JsonConvert.DeserializeObject(requestBody);
16     name = name ?? data?.name;
17
18     string responseMessage = string.IsNullOrEmpty(name)
19     ? "This HTTP triggered function executed successfully. Pass a name in the query string or in the request body for a personalized response."
20     : $"Hello, {name}. This HTTP triggered function executed successfully.";
21
22     return new OkObjectResult(responseMessage);
23 }
```

Logs

The screenshot shows the Microsoft Azure Functions blade for a function named 'HttpTrigger1'. The code editor displays the following C# code:

```

1  #r "Newtonsoft.Json"
2
3  using System;
4  using Microsoft.AspNetCore.Mvc;
5  using Microsoft.Extensions.Primitives;
6  using Newtonsoft.Json;
7
8  public static async Task<ActionResult> Run(HttpContext req, ILogger log)
9  {
10     string response = JsonConvert.SerializeObject("Hello Sahil! Serverless API Connected with AZURE function");
11 }

```

The logs panel below shows the output: 'Connected!'

Kindly copy the Function URL, and visit [Workspaces \(postman.co\)](#)

The screenshot shows the Microsoft Azure Functions blade for 'HttpTrigger1'. A modal dialog titled 'Get function URL' is open, showing the key 'default' and the URL 'https://cad10-practical-sahil.azurewebsites.net/api/HttpTrigger1'. The 'Copy to clipboard' button is highlighted.

Paste the URL Link in “GET” to trigger our server less API:

The screenshot shows a Postman workspace with a new GET request. The URL is 'https://web.postman.co/workspace/0a74b0ca-47e0-408d-8b10-149f6b47f2ad/request/14389326-66574cf3-ff1a-4cde-ae5f-86e7b03c5120'. The request is set to 'GET' and has a 'Query Params' section with a single entry: 'Key' (Value: 'Value'). The 'Send' button is highlighted.

Click on Send:

The screenshot shows the Postman application interface. On the left, there's a sidebar with options like Home, Workspaces, Reports, Explore, Collections, APIs, Environments (which is selected), Mock Servers, Monitors, and History. The main area shows a workspace named 'CAD_Practical'. A GET request is being prepared to the URL `https://cad10-practical-sahil.azurewebsites.net/api/HttpTrigger1?code=ay5Zfd5tsAbc3W6BBalAcv8d8ePQjVGvMGlv1F7TY0InnfGGyYzAw==`. The 'Pre-request Script' tab is selected, containing the number '1'. To the right, there's a snippet of JavaScript code: 'Pre-request scripts are written in JavaScript, and are run before the request is sent.' Below it are links to 'Learn more about pre-request scripts', 'SNIPPETS', and several environment variable-related functions: 'Get an environment variable', 'Get a global variable', 'Get a variable', 'Set an environment variable', and 'Set a global variable'. At the bottom, a progress bar says 'Sending request...' with a 'Cancel' button. A note says 'Hit Send to get a response.'

We will see our API being executed:

This screenshot shows the same Postman interface after the request has been sent. The status bar at the bottom indicates 'Status: 200 OK' and 'Time: 1807 ms'. The response body is displayed as a single line of text: 'Hello Sahil! Serverless API Connected with AZURE function'. The rest of the interface is identical to the first screenshot, showing the workspace, request details, and available snippets.

Practical no 11

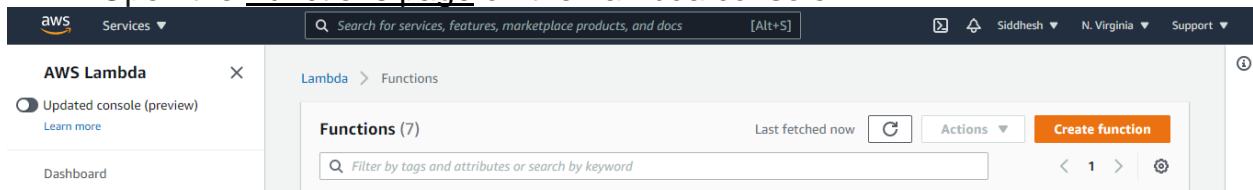
Create AWS Lambda Function

Create AWS Lambda function:

To create a Lambda function with the console

Step 1:

Open the [Functions page](#) on the Lambda console.



Step 2:

Choose **Create function**.



Step 3:

Under **Basic information**, do the following:

- a. For **Function name**, enter **mypractcad11**
- b. For **Runtime**, confirm that **python 3.8** is selected.

Step 4:

Choose **Create function**



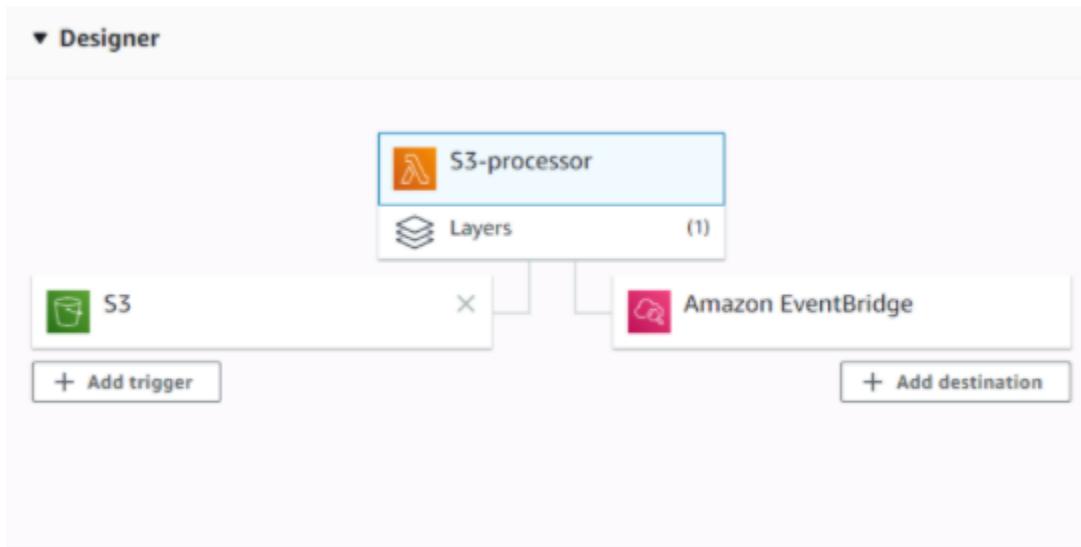
Step 5:

Lambda creates a **python3.8** function and an [execution role](#) that grants the function permission to upload logs. The Lambda function assumes the execution role when you

invoke your function, and uses the execution role to create credentials for the AWS SDK and to read data from event sources.

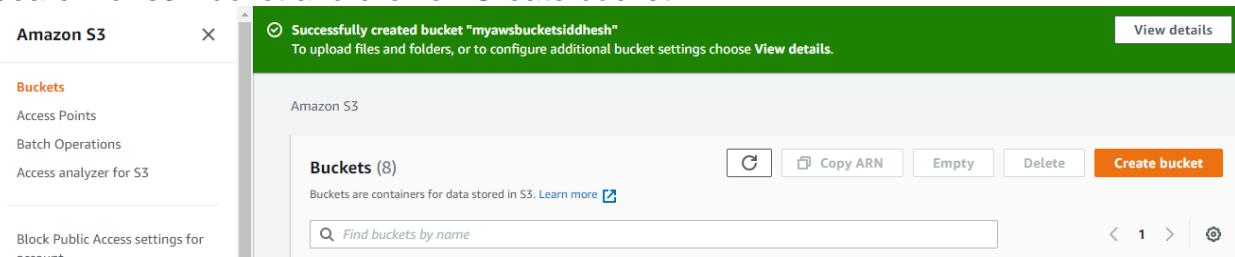
Step 5:

Use the designer: For Create the S3 Bucket



Step 6:

Search for s3 Bucket and click on Create bucket.



Step 7:

Amazon S3 > Create bucket

Create bucket

Buckets are containers for data stored in S3. [Learn more](#)

General configuration

Bucket name
 Bucket name must be unique and must not contain spaces or uppercase letters. [See rules for bucket naming](#)

Region

Copy settings from existing bucket - *optional*
Only the bucket settings in the following configuration are copied.
[Choose bucket](#)

Step 8:
Search for IAM- Identity and Access Management

Click on Roles leftside & selected Function which created earlier

Permissions Trust relationships Tags Access Advisor Revoke sessions

▼ Permissions policies (2 policies applied)

[Attach policies](#) [+ Add inline policy](#)

| Policy name ▾ | Policy type ▾ | X |
|---|----------------|---|
| AWSLambdaBasicExecutionRole-e9c48466-622b-4701-9d2d-e3e878dd... | Managed policy | X |

Step 9:
Give permission to that function “[AmazonS3FullAccess](#)” if we will not give that acces it’s shows an error

Add permissions to mypraccad11-role-da87xg9d

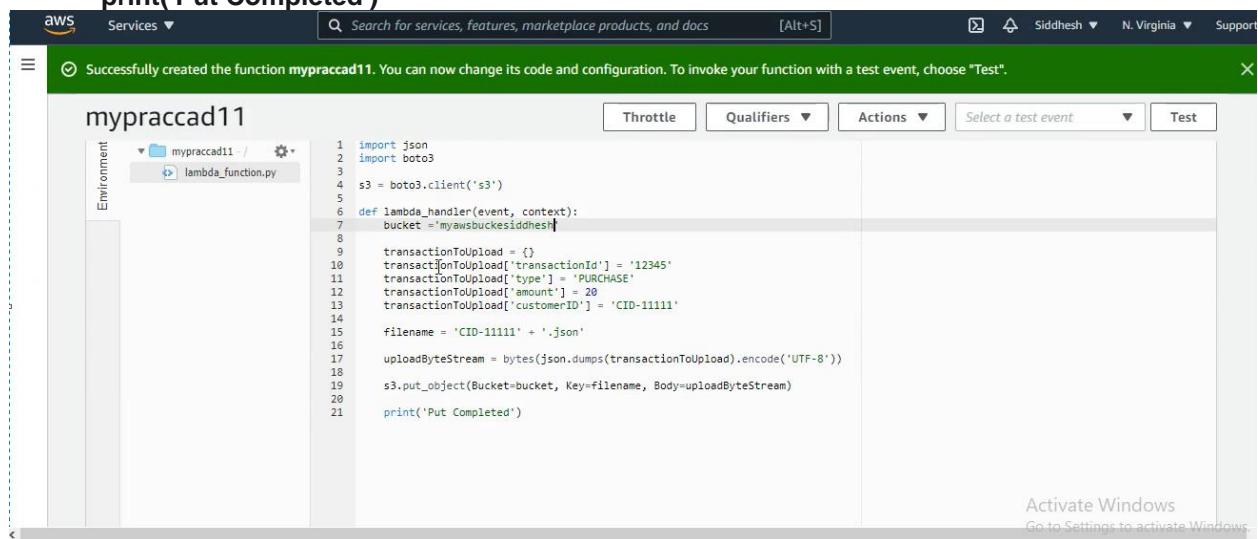
Attach Permissions

| Policy name | | Type | Used as |
|-------------------------------------|---|-------------|------------------------|
| <input type="checkbox"/> | AmazonDMSRedshiftS3Role | AWS managed | None |
| <input checked="" type="checkbox"/> | AmazonS3FullAccess | AWS managed | Permissions policy (4) |
| <input type="checkbox"/> | AmazonS3OutpostsFullAccess | AWS managed | None |
| <input type="checkbox"/> | AmazonS3OutpostsReadOnlyAccess | AWS managed | None |
| <input type="checkbox"/> | AmazonS3ReadOnlyAccess | AWS managed | None |
| <input type="checkbox"/> | QuickSightAccessForS3StorageManagementAnalyticsReadOnly | AWS managed | None |

Step 10:

Type our Code in Lambda.py console

```
import json
import boto3
s3 = boto3.client('s3')
def lambda_handler(event, context):
    bucket ='vikascollegedemo'
    transactionToUpload = {}
    transactionToUpload['transactionId'] = '12345'
    transactionToUpload['type'] = 'PURCHASE'
    transactionToUpload['amount'] = 20
    transactionToUpload['customerID'] = 'CID-11111'
    filename = 'CID-11111' + '.json'
    uploadByteStream = bytes(json.dumps(transactionToUpload).encode('UTF-8'))
    s3.put_object(Bucket=bucket, Key=filename, Body=uploadByteStream)
    print('Put Completed')
```



The screenshot shows the AWS Lambda console interface. At the top, there's a search bar and navigation links for AWS services, regions (Siddhesh, N. Virginia), and support. Below the header, a message says "Successfully created the function mypraccad11. You can now change its code and configuration. To invoke your function with a test event, choose "Test".". The main area displays the Lambda function details for "mypraccad11". On the left, there's a sidebar with "Environment" settings. The function configuration includes a "Throttle" section, "Qualifiers" dropdown, "Actions" dropdown, "Select a test event" dropdown, and a "Test" button. The code editor on the right contains the Python script provided above. The code uses the boto3 library to interact with the S3 service, creating a new object named 'CID-11111.json' containing a purchase transaction with ID '12345', type 'PURCHASE', amount 20, and customer ID 'CID-11111'. The AWS logo and copyright information are visible at the bottom.

Step 11:

After deploying SourceCode Click on “Test Button” and output will be shown

Execution result: succeeded (logs)

▼ Details

The area below shows the result returned by your function execution. [Learn more](#) about returning results from your function.

```
null
```

Output:

Log output

The section below shows the logging calls in your code. [Click here](#) to view the corresponding CloudWatch log group.

```
START RequestId: c78bd4a8-9880-4dec-af4c-8bb644b2b2ab Version: $LATEST
Put Completed
END RequestId: c78bd4a8-9880-4dec-af4c-8bb644b2b2ab
REPORT RequestId: c78bd4a8-9880-4dec-af4c-8bb644b2b2ab Duration: 257.69 ms Billed Duration: 258 ms Memory Size: 128 MB Max Memory Used: 78 MB Init Duration: 379.57 ms
```

mypraccad11

Throttle Qualifiers ▾ Actions ▾ hello ▾ Test

Execution result: succeeded (logs)

▼ Details

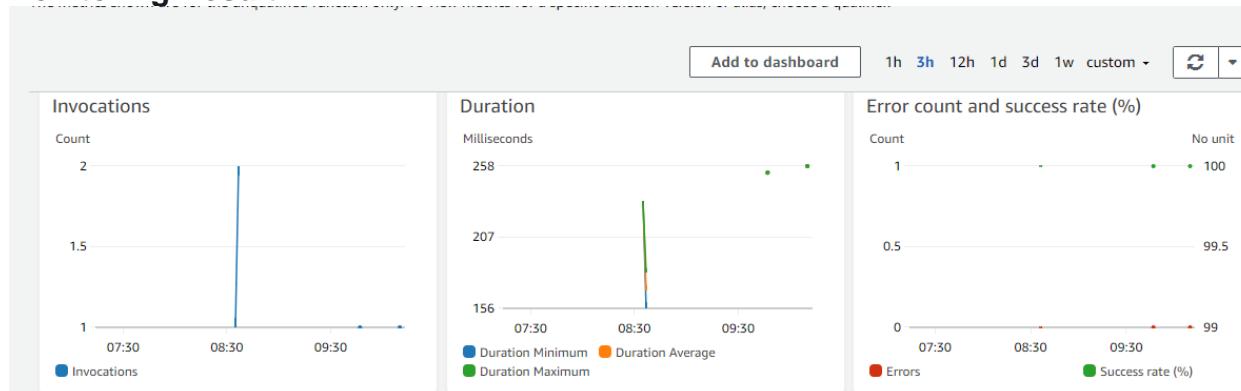
The area below shows the result returned by your function execution. [Learn more](#) about returning results from your function.

```
null
```

Summary

| | |
|--|--------------------------------------|
| Code SHA-256 | Request ID |
| zTyynrHbSPuiQEC4arOmDvHKi41IkMMund1nNU2OEpw= | c78bd4a8-9880-4dec-af4c-8bb644b2b2ab |
| Init duration | Duration |
| 379.57 ms | 257.69 ms |
| Billed duration | Resources configured |
| 258 ms | 128 MB |

Monitoring Result:



Execution Result:

| Execution result | |
|----------------------|--|
| ▼ Execution results | Status: Succeeded Max memory used: 78 MB Time: 257.69 ms |
| Response | null |
| Function Logs | START RequestId: c78bd4a8-9880-4dec-af4c-8bb644b2b2ab Version: \$LATEST Put Completed END RequestId: c78bd4a8-9880-4dec-af4c-8bb644b2b2ab REPORT RequestId: c78bd4a8-9880-4dec-af4c-8bb644b2b2ab Duration: 257.69 ms Billed Duration: 258 ms Memory Size: 128 MB Max Memory Used: 78 MB |
| Request ID | c78bd4a8-9880-4dec-af4c-8bb644b2b2ab |

Practical no 12

Build AWS Lambda using API Gateway

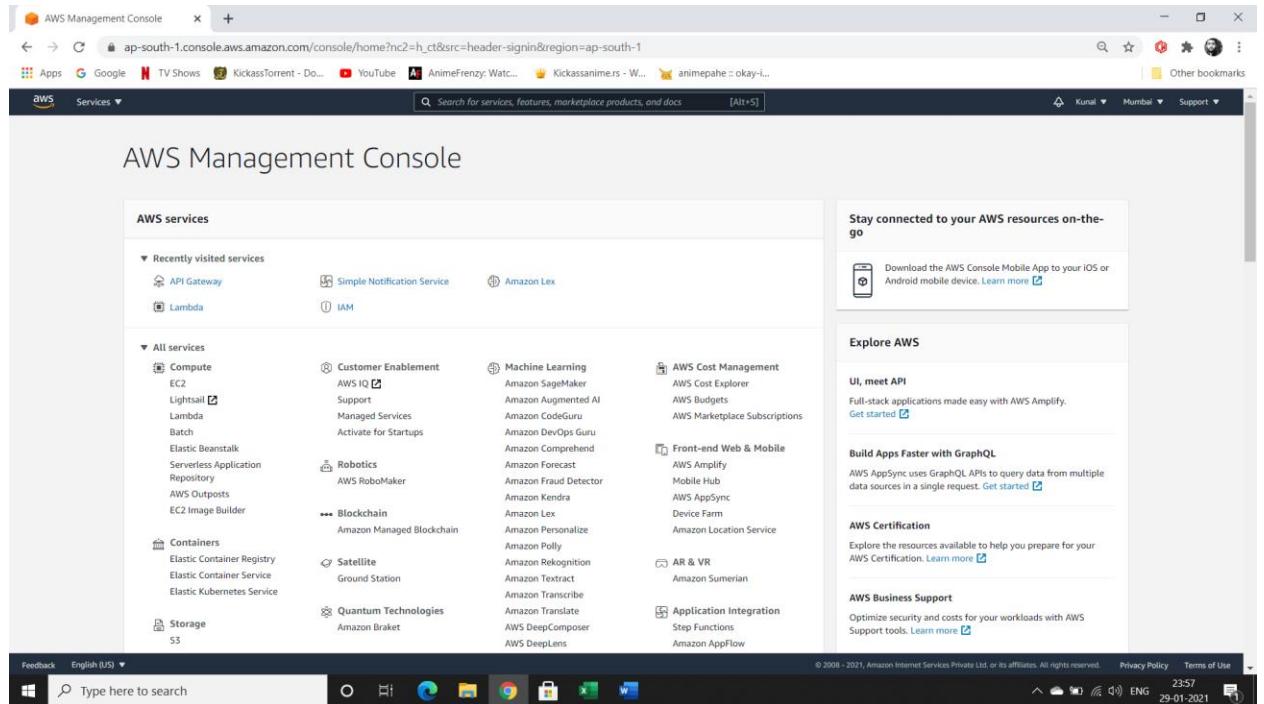
Description of Practical: -

- In this practical we will learn to build AWS Lambda function using API Gateway.
- In order to perform this practical we will take an example.
- The example is EMI Calculator.

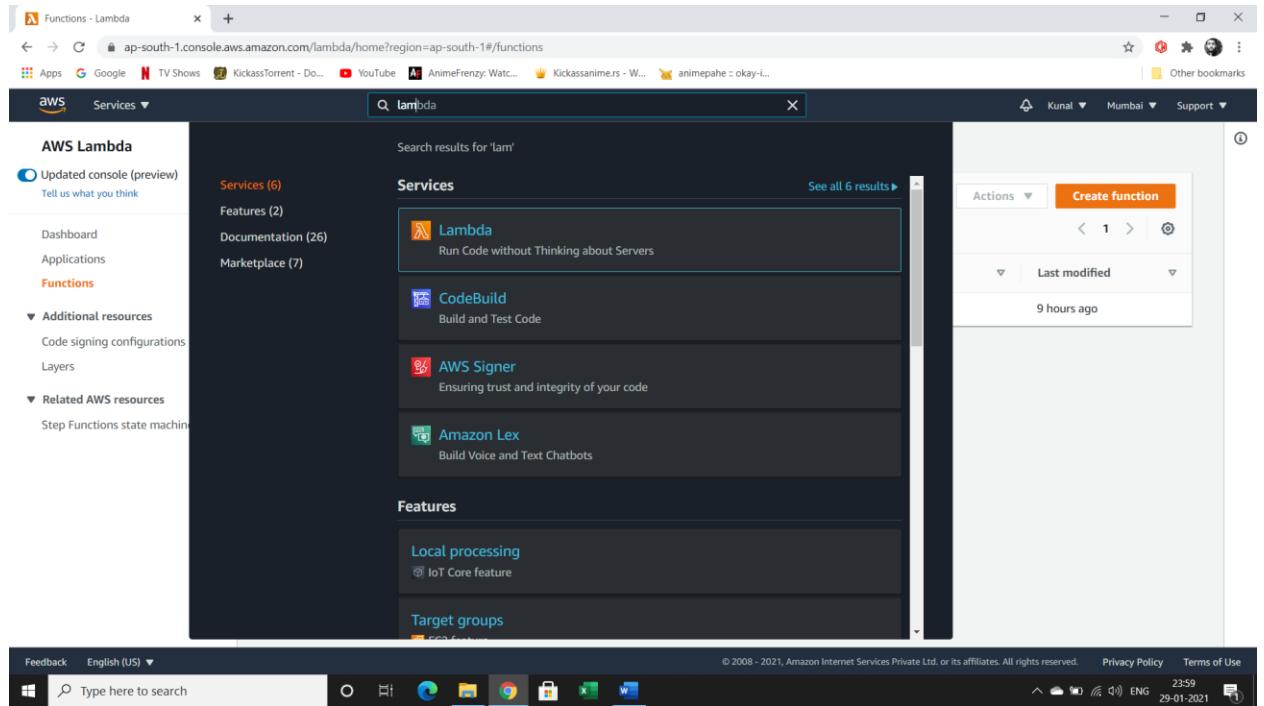
EMI Calculator

Steps

- Login into AWS Console.
- Search for Lambda Function on the service bar.

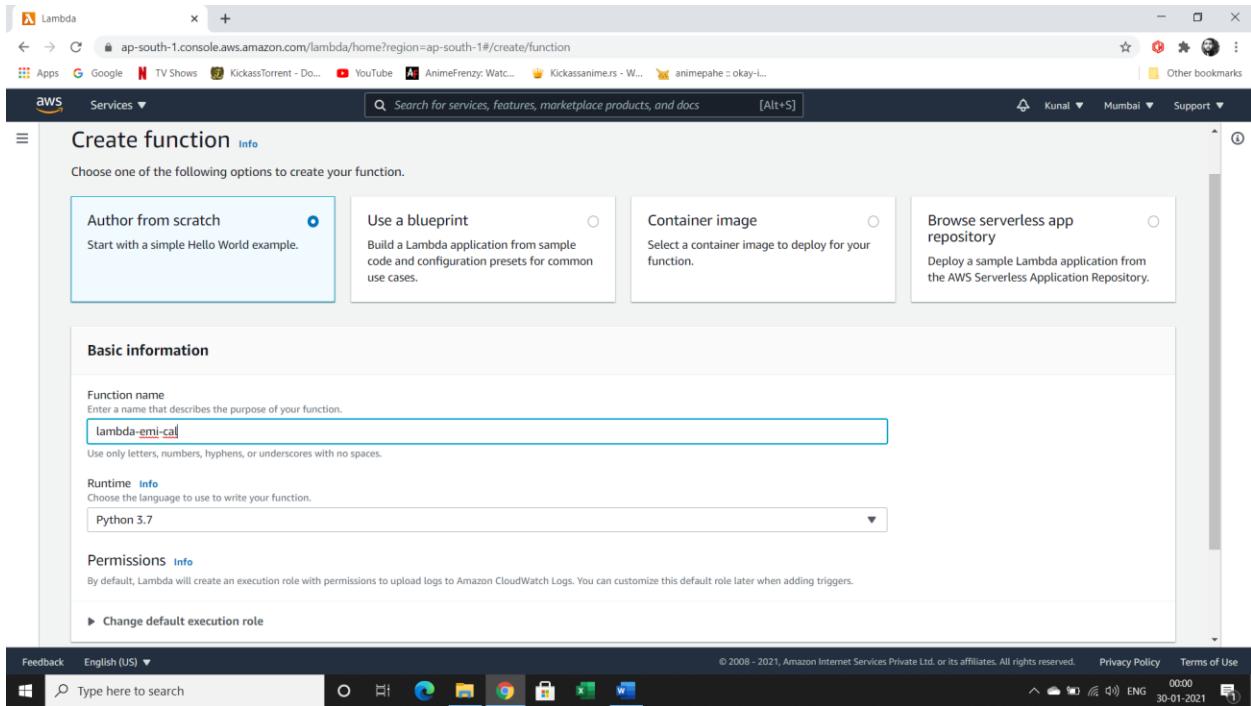


- Here we will create a lambda function for the practical.

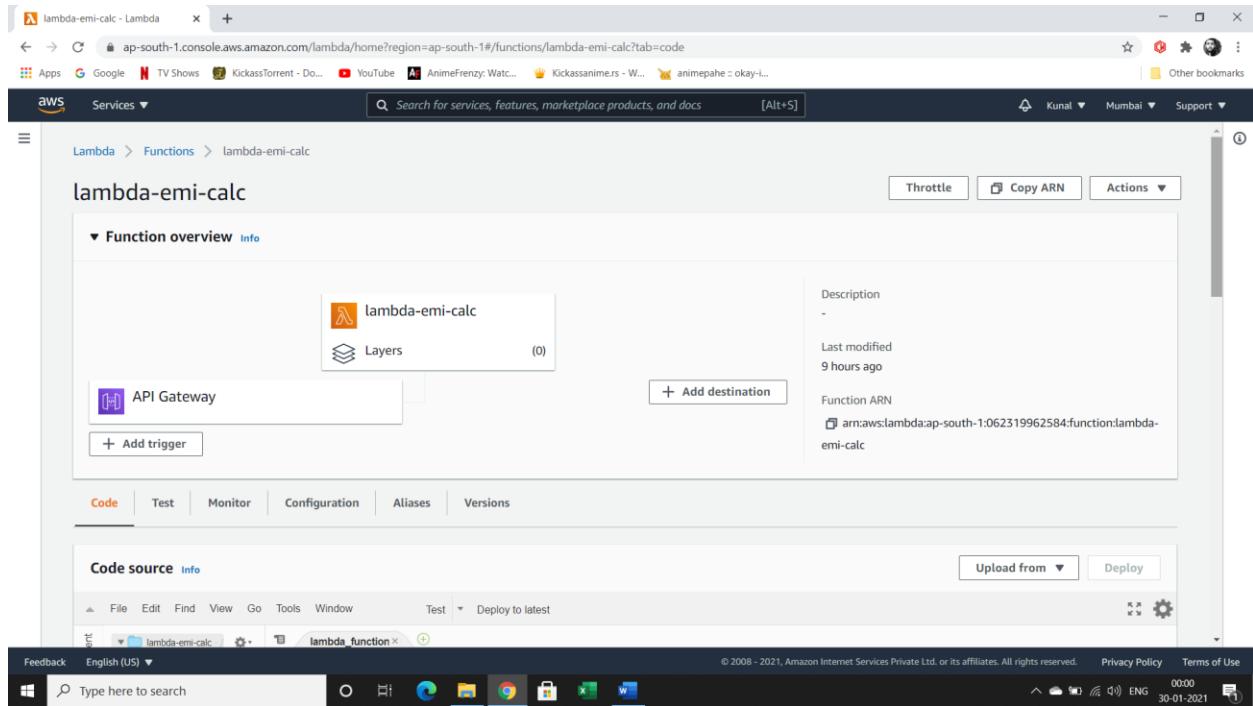


- Click on Create Function on the Lambda Tab.

- Here we will choose Author from Scratch.

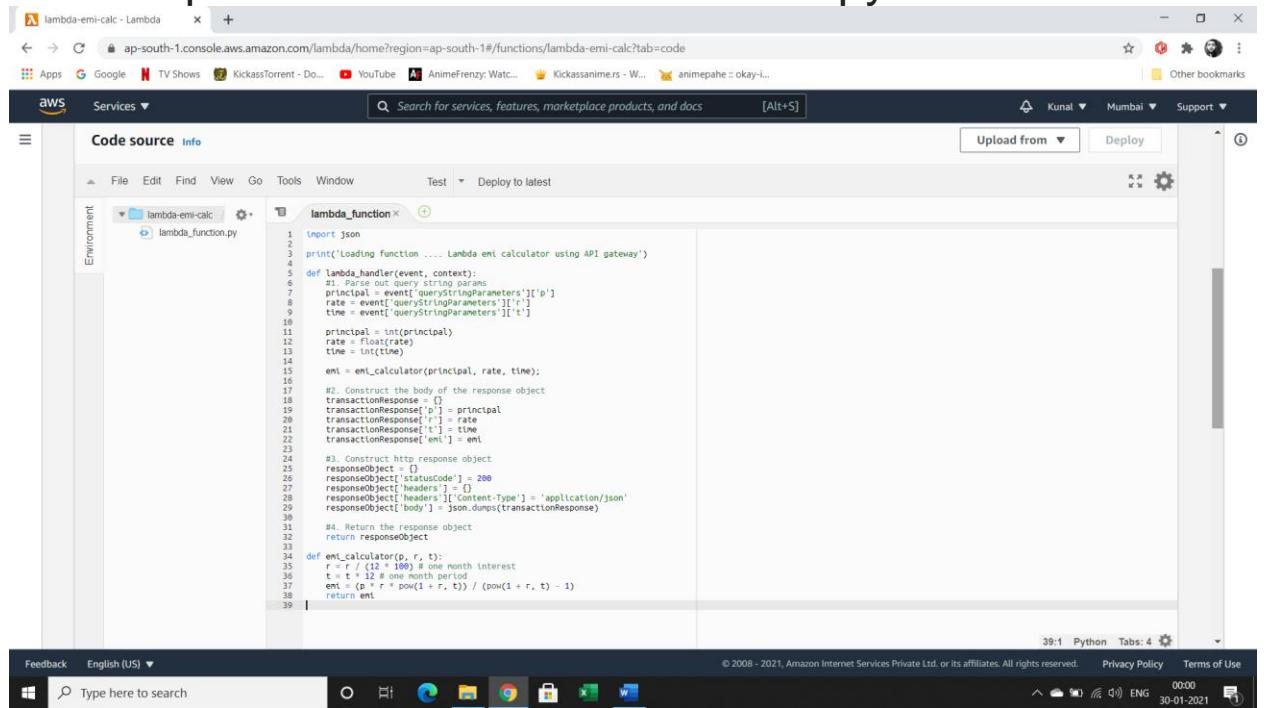


- After selecting that write the function name as lambda-emi-calc
 - We will be running the code on python 3.7
 - So, we will choose runtime as python 3.7.
 - After that click on create function.
-
- We will see that it has created lambda function with basic default code.



- We can see that it has created lambda function with basic default code

- Next step is to enter the emi calculator python code



```

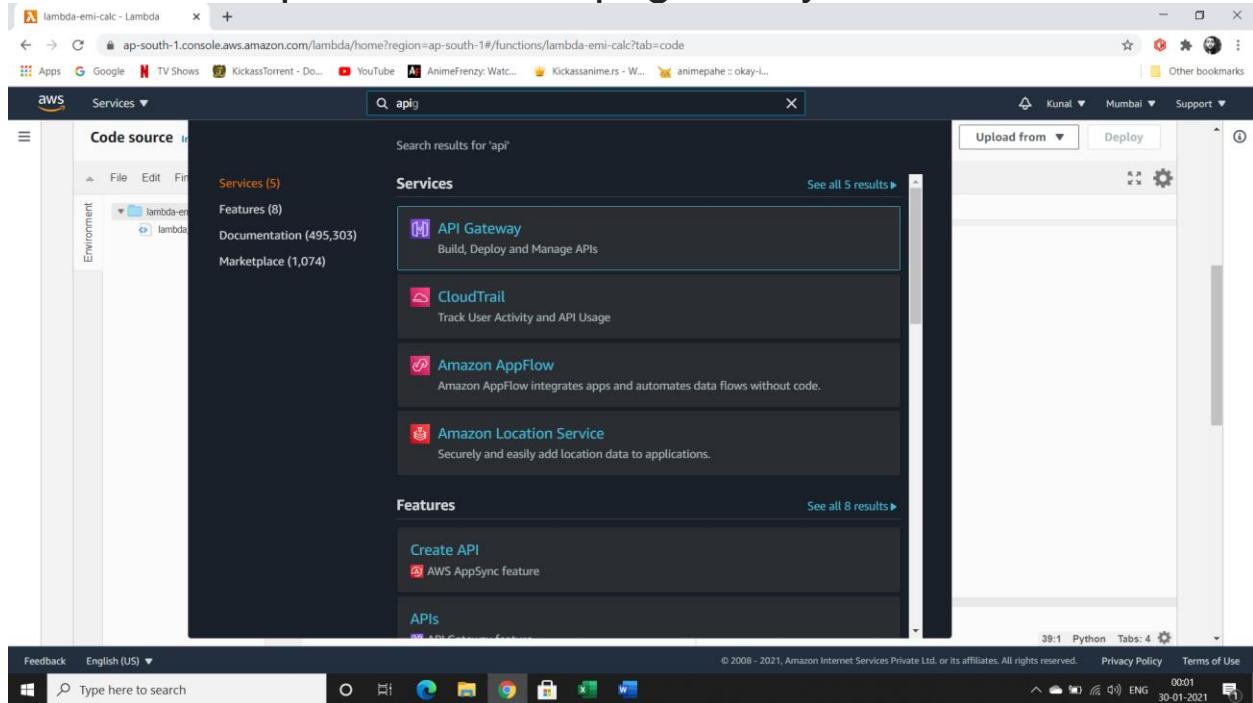
1 import json
2
3 print('Loading function .... Lambda emi calculator using API gateway')
4
5 def lambda_handler(event, context):
6     # Parse out query string parameters
7     principal = event['queryStringParameters']['p']
8     rate = event['queryStringParameters']['r']
9     time = event['queryStringParameters']['t']
10
11     principal = int(principal)
12     rate = float(rate)
13     time = int(time)
14
15     emi = emi_calculator(principal, rate, time);
16
17     # Construct the body of the response object
18     transactionResponse = {}
19     transactionResponse['p'] = principal
20     transactionResponse['r'] = rate
21     transactionResponse['t'] = time
22     transactionResponse['emi'] = emi
23
24     # Construct http response object
25     responseObject = {}
26     responseObject['statusCode'] = 200
27     responseObject['headers'] = {}
28     responseObject['headers']['Content-Type'] = 'application/json'
29     responseObject['body'] = json.dumps(transactionResponse)
30
31     # Return the response object
32     return responseObject
33
34 def emi_calculator(p, r, t):
35     r = r / (12 * 100) # one month interest
36     t = t * 12 # one month period
37     emi = (p * r * pow(1 + r, t)) / (pow(1 + r, t) - 1)
38
39

```

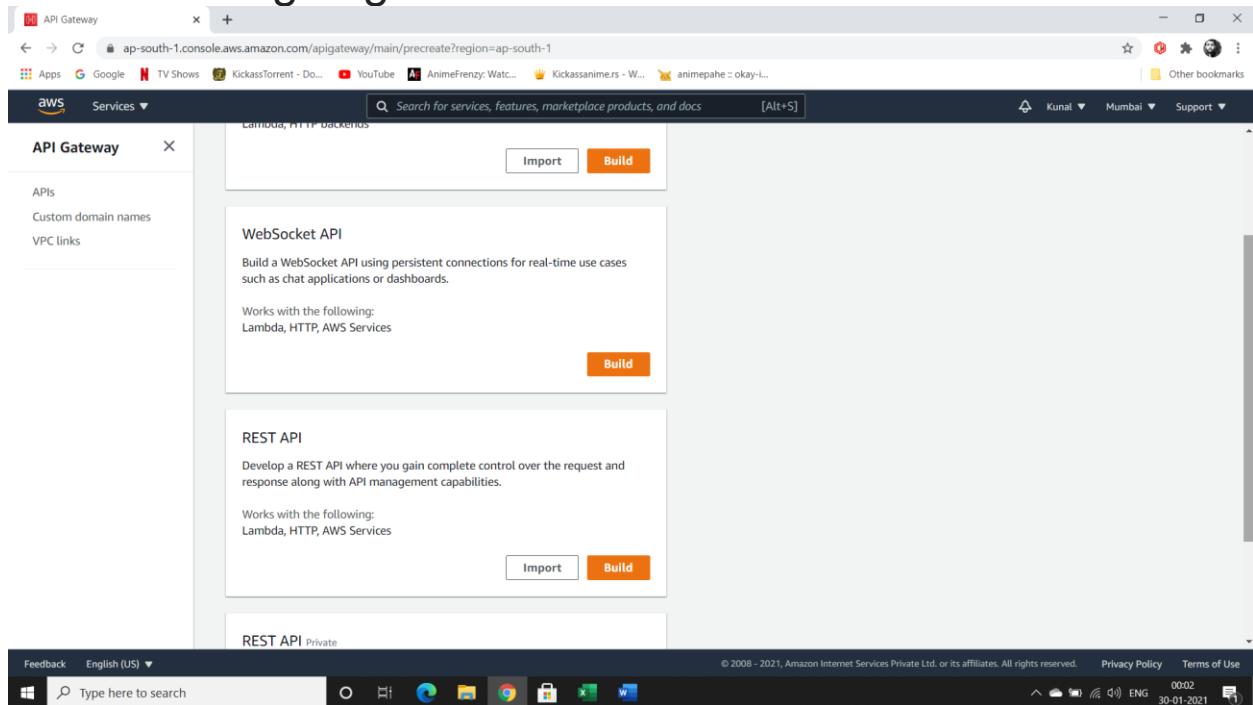
The screenshot shows the AWS Lambda console interface. The top navigation bar includes tabs for 'Code source' and 'Info'. The main area displays the Python code for a lambda function named 'lambda-emi-calc'. The code defines a lambda handler that takes query string parameters for principal, rate, and time, calculates the EMI using a provided function, and returns a JSON response. The AWS Lambda logo is visible in the top left corner of the browser window.

- Save the lambda function
- We are not defining any environment variable
- Select basic setting and choose runtime as python 3.7

- Next step is to create api gateway



- We are going to choose REST API



- Click on new API

| Name | Description | ID | Protocol | Endpoint type | Created |
|------------|-------------|------------|----------|---------------|------------|
| EmiCalcApi | EmiCalcApi | k5qya2hzql | REST | Regional | 2021-01-29 |

- In the settings tab do the following changes
- API name = EmiCalcApi
- Description = EmiCalcApi
- Endpoint type = Regional
- Click on create tab

- Next step click on action then create resource

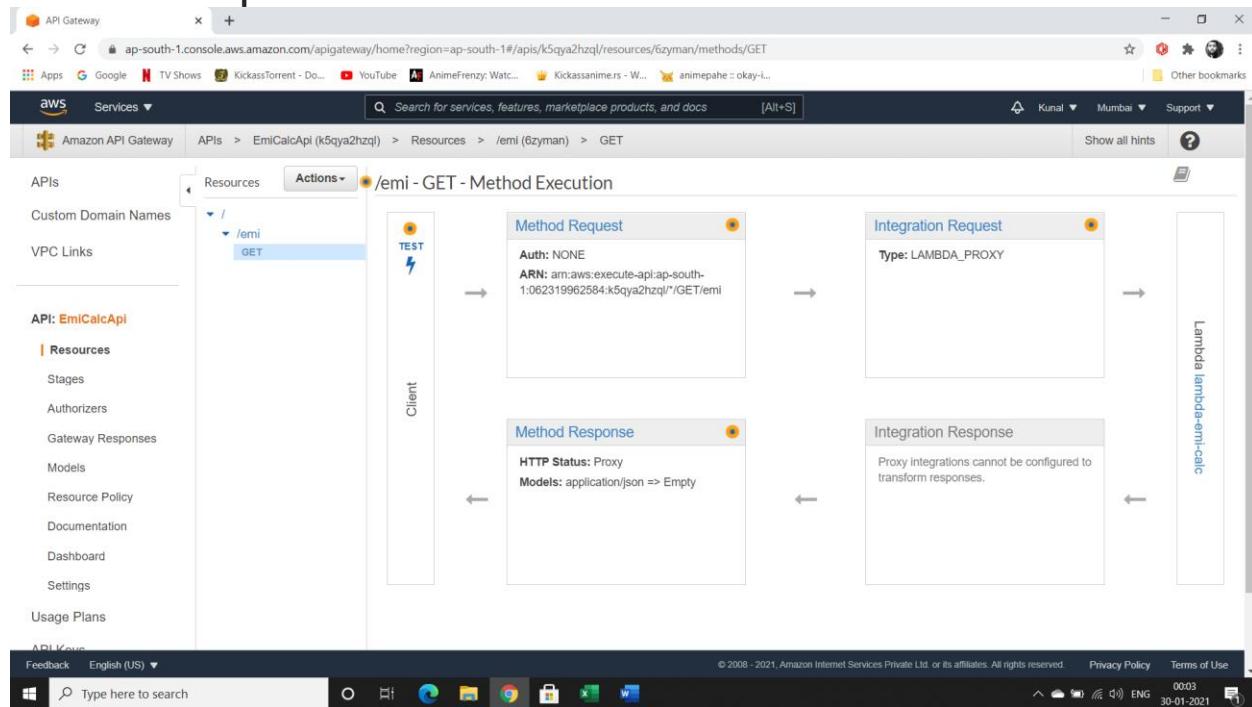
The screenshot shows the AWS API Gateway console. The URL is ap-south-1.console.aws.amazon.com/apigateway/home?region=ap-south-1#/apis/k5qya2hzql/resources/bsbjjejnc9. The left sidebar shows the navigation path: APIs > EmiCalcApi (k5qya2hzql) > Resources > / (bsbjjejnc9). The main area displays the 'Actions' dropdown menu for the resource path '/'. The 'RESOURCE ACTIONS' section includes: Create Method, Create Resource, Enable CORS, and Edit Resource Documentation. The 'API ACTIONS' section includes: Deploy API, Import API, Edit API Documentation, and Delete API. A message at the bottom right states: 'No methods defined for the resource.'

- Enter resource name as emi

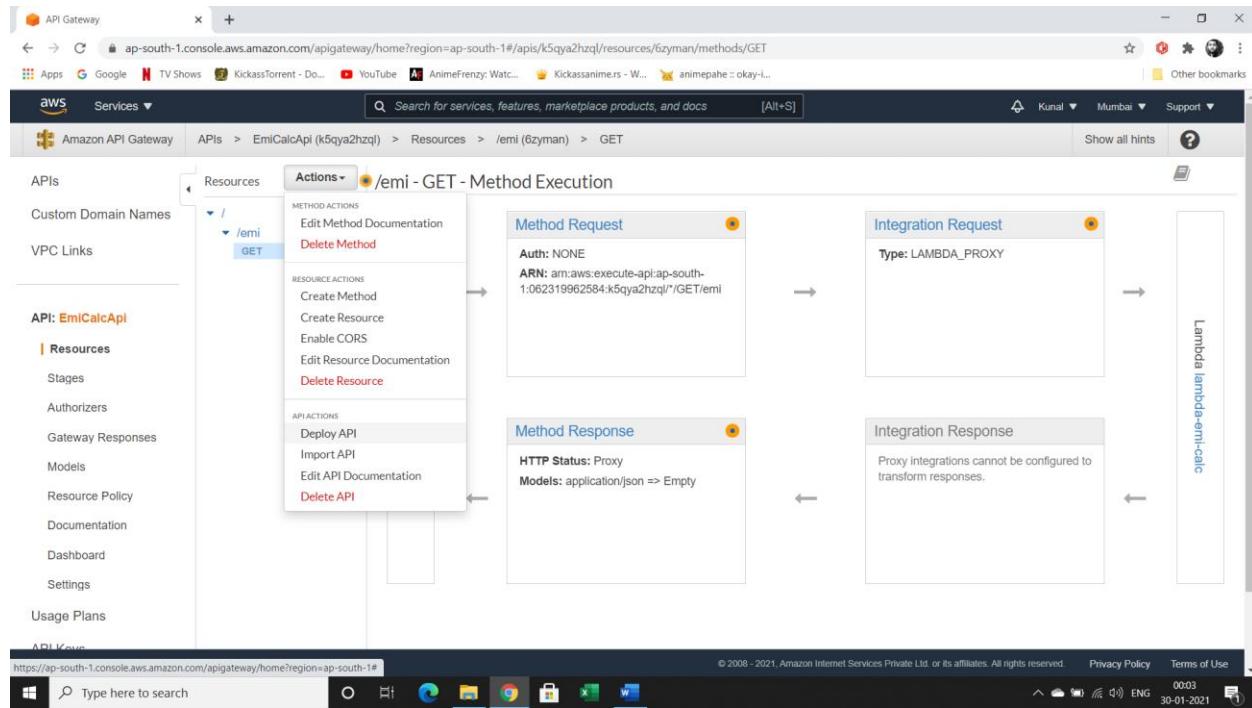
- And create the resource

The screenshot shows the AWS API Gateway console with the URL ap-south-1.console.aws.amazon.com/apigateway/home?region=ap-south-1#/apis/k5qya2hzql/resources/bsbjjejnc9/create. The left sidebar shows the navigation path: APIs > EmiCalcApi (k5qya2hzql) > Resources > / (bsbjjejnc9) > Create. The main area displays a 'New Child Resource' dialog. It shows the path configuration: 'Configure as proxy resource' (unchecked), 'Resource Name*' (set to 'My Resource'), and 'Resource Path*' (set to '/my-resource'). A note explains: 'You can add path parameters using brackets. For example, the resource path {username} represents a path parameter called 'username'. Configuring /{proxy+} as a proxy resource catches all requests to its sub-resources. For example, it works for a GET request to /foo. To handle requests to /, add a new ANY method on the / resource.' At the bottom right are 'Cancel' and 'Create Resource' buttons.

- Now select on action tab and create method
- Use get parameter
- We are using integration type as Lambda Function
- Enter lambda function as lambda-emi-calc
- Save the changes
- Grant permission for lambda function

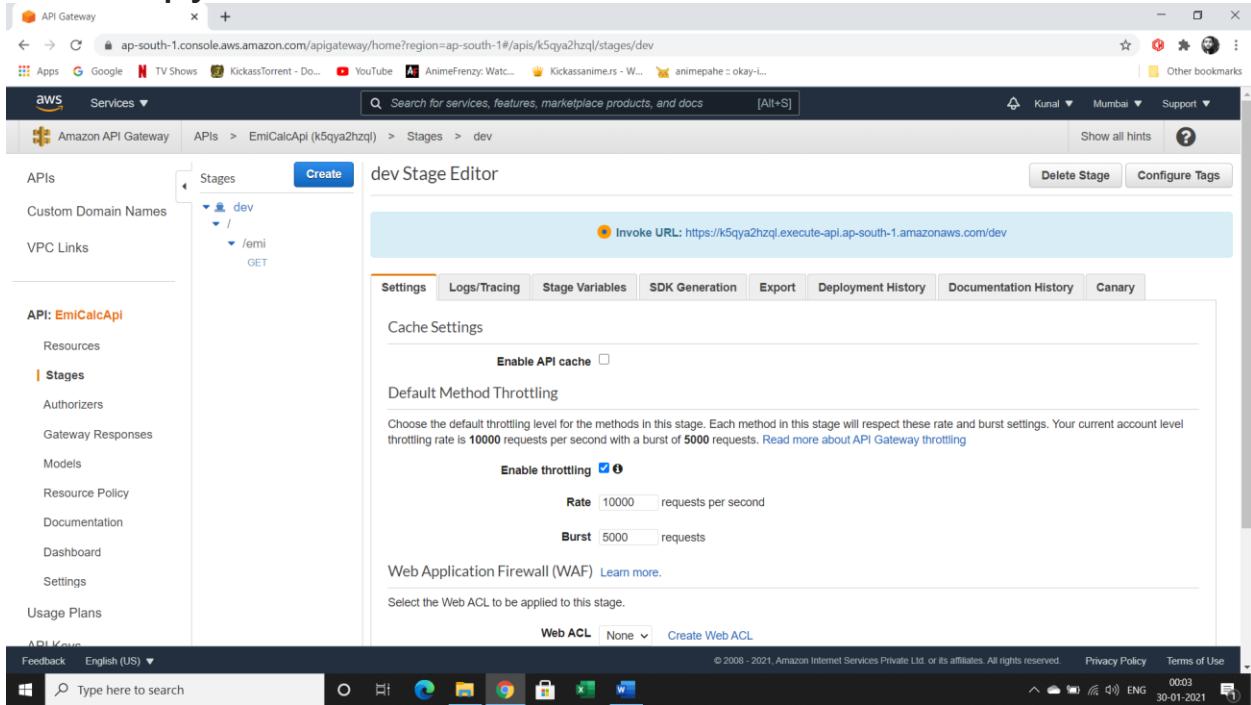


- Next step is to click on action and then deploy api



- Enter deployment stage as New Stage
- Enter stage name as dev
- Stage description as dev
- Deployment description as dev
- After that click on GET method

- Copy the URL



- To calculate the EMI we have to place valid parameters

- We have 3 parameters

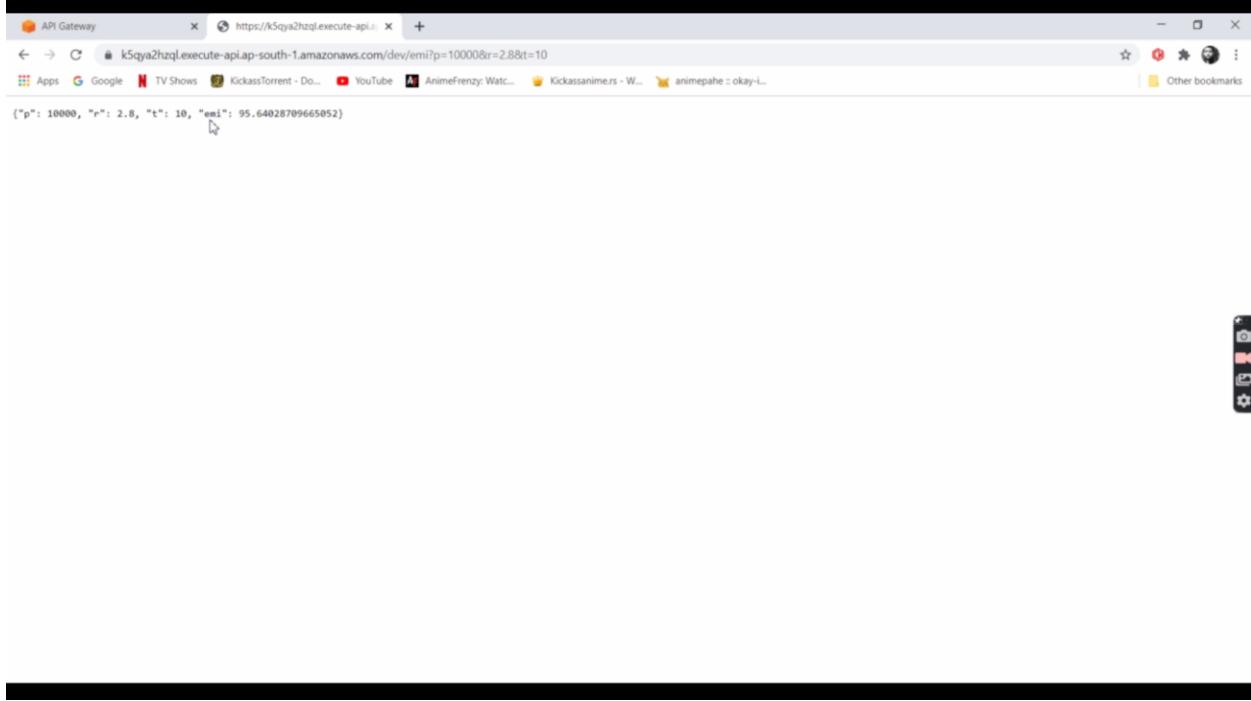
Principle = p

Rate = r

Time = t

- Add the follow to the copied url
- ?p= certain value for eg 10000
- Then add rate (r) as certain value 2.8 and the final value time as 10

- `emi?p=10000&r=2.8&t=10`
- Click on enter and you will have your required emi



Python Code for EMI Calculator

```
"Untitled - Notepad
File Edit Format View Help
import json

print('Loading function .... Lambda emi calculator using API gateway')

def lambda_handler(event, context):

    principal = event['queryStringParameters']['p']
    rate = event['queryStringParameters']['r']
    time = event['queryStringParameters']['t']

    principal = int(principal)
    rate = float(rate)
    time = int(time)

    emi = emi_calculator(principal, rate, time);

    transactionResponse = {}
    transactionResponse['p'] = principal
    transactionResponse['r'] = rate
    transactionResponse['t'] = time
    transactionResponse['emi'] = emi

    responseObject = {}
    responseObject['statusCode'] = 200
    responseObject['headers'] = {}
    responseObject['headers'][['Content-Type']] = 'application/json'
    responseObject['body'] = json.dumps(transactionResponse)

    return responseObject

def emi_calculator(p, r, t):
    r = r / (12 * 100)
    t = t * 12
    emi = (p * r * pow(1 + r, t)) / (pow(1 + r, t) - 1)
    return emi
|
```

